```java
/**
 *
 */
//@author maximilian raspe
package snake;
/**
 * @author  Sven Loeffler
 * @date    04.12.2018
 * @version 1.0
 */
public class Main {
   public static void main(String[] args) {
      GameBoard board = new GameBoard();
      board.startGame();
   }

}

//@author maximilian raspe
package snake;
public abstract class Snake implements ISnake {
    private char dir;
    private int length;
    private GameBoard board;
    private Body snake;
    /**
     * Creates a new Snake with default length in the middle of the Console.
     * Every body cell has the default position for x and y.
     *
     * The default direction should be GO_RIGHT.
     *
     * @param board
     * Set the board of the snake to board.
     */
    public Snake(GameBoard board) {
        for (int i = 0; i < DEFAULT_START_LENGTH; i++) {
            body.add(new Body(10, 10));
        }
        this.dir = GO_RIGHT;
        this.board = board;
    }
    //setter methode für die richtung und jeweiligen tastendruck
    @Override
    public void setDir(char dir) {
        if(dir == 'A') {
            this.dir = GO_LEFT;
        }
        if(dir == 'W') {
            this.dir = GO_UP;
        }
        if(dir == 'D') {
            this.dir = GO_RIGHT;
        }
        if(dir == 'S') {
            this.dir = GO_DOWN;
        }
    }
    //getter methode für direction
    @Override
    public char getDir() {
        return this.dir;
    }
    //länge der liste
    @Override
    public int getLength() {
        return body.size();
    }
    //prüft ob kopf der schlange an einem apfel ist
    @Override
    public boolean isEating() {
        if(snake.equals(board.getAppleBody())) {
            return true;
        }
        else return false;
    }
    @Override
    public boolean move() {
        int x = snake.getPositionX();
        int y = snake.getPositionY();
        if(isEating() == true) {
            body.addLast(snake);
            board.setApple();
        }
        if(isSelfEating() == true) {
            return false;
        }
        if(leavesTheBoard() == true) {
            return false;
        }
        if(getDir() == GO_LEFT) {
            snake.setPositionX(x - 1);
            ISnake.body.addLast(snake);
            ISnake.body.removeFirst();
        }
        if(getDir() == GO_RIGHT) {
            snake.setPositionX(x + 1);
            ISnake.body.addLast(snake);
            ISnake.body.removeFirst();
        }
        if(getDir() == GO_UP) {
            snake.setPositionY(y + 1);
            ISnake.body.addLast(snake);
            ISnake. body.removeFirst();
        }
        if(getDir() == GO_DOWN) {
           snake.setPositionY(y - 1);
           ISnake.body.addLast(snake);
           ISnake.body.removeFirst();
        }
        return true;
    }
    @Override
    public boolean leavesTheBoard() {
        if(snake.getPositionX() > 40) {
            return true;
        }
        if(snake.getPositionX() < 0) {
            return true;
        }
        if(snake.getPositionY() > 27) {
            return true;
        }
        if(snake.getPositionY() < 0) {
            return true;
        }
        return false;
    }
    @Override
    public boolean isSelfEating() {
        for(int i = 1; i <= body.size(); i++) {
            if (ISnake.body.getFirst().equals(ISnake.body.get(i))) {
                return true;
            }
        }
        return false;
    }
}

/**@author maximilian raspe
 *
 */
package snake;
import java.util.LinkedList;
/**
 * @author Viktoria, Sven
 * @date 03.12.2018
 * @version 1.0
 *
 *            The snake that snakes over the gameboard with attributes length,
 *            direction and body which is an ArrayList of snake.Body elements.
 */
public interface ISnake {
   /**
    * Possible directions of the snake: go up
    */
   final static char GO_UP = 'w';
   /**
    * Possible directions of the snake: go down
    */
   final static char GO_DOWN = 's';
   /**
    * Possible directions of the snake: go left
    */
   final static char GO_LEFT = 'a';
   /**
    * Possible directions of the snake: go right
    */
   final static char GO_RIGHT = 'd';
   /**
    * The default x and y position of a body cell.
    */
   final static int DEFAULT_BODY_POSITION = 0;
   /**
    * The defulat start length of the snake.
    */
   final static int DEFAULT_START_LENGTH = 3;
   /**
    * The body list contains body elements which have knowledge about there x
    * and y position.
    */
   LinkedList<Body> body = new LinkedList<Body>();
   /**
    * Check if the given char is in { GO_UP, GO_DOWN, GO_LEFT, GO_RIGHT} then
    * it set this.dir to the given value otherwise do nothing.
    *
    * @param dir
    *            The new direction of the snake.
    */
   public void setDir(char dir);
   /**
```

```
     * @return The direction of the snake.
     */
    public char getDir();
    /**
     * Returns true if the position of the head of the snake is equal to the
     * position of the apple from the game board.
     *
     * @return true if the head of the snake have the same position like the
     *         apple otherwise false.
     */
    public boolean isEating();
    /**
     * Returns the number of body cells.
     *
     * @return The number of body cells.
     */
    public int getLength();
    /**
     * Change the coordinates of the first body cell to old coordinates plus the
     * step into its direction. Set the coordinates of every other cell to the
     * coordinates of the previous body cell.
     *
     * Call the isEating method and add a body cell to the body if necessary and
     * set a new apple if necessary.
     *
     * Check if the snake is self eating or leave the board.
     *
     * @return True if the snake does not leave the game or is self eating
     *         otherwise false.
     */
    public boolean move();
    /**
     * Check if the snake is eating itself. A snake is eating itself, if the
     * first body position is equal to any other body position.
     *
     * @return True if the snake eat itself, false otherwise.
     */
    public boolean isSelfEating();
    /**
     * Check if the snake leaves the board or not.
     *
     * @return True if the snake leaves the board otherwise false.
     */
    public boolean leavesTheBoard();
}

/**
 *
 *///@author maximilian raspe
package snake;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Timer;
/**
 * @author Sven Loeffler
 * @date 04.12.2018
 * @version 1.0
 */
public class GameBoard {
    private static Body appleBody;
    private static ISnake snake;
    private GameBoard board;
```

```
   /**
    * The timer which handle the inputs.
    */
   private static Timer timer;
   private static int time = 100;
   /**
    * Creates a new GameBoard. For this, a new Snake is created with this
    * GameBoard as input, then a new appleBody is created and the apple is set.
    */
   public GameBoard() {
      Console.setBackground(Color.black);    //malt konsole und generiert apfel
      Console.setForeground(Color.white);
      appleBody = new Body(5, 7);
      drawApple();
      drawSnake();
   }
   /**
    * Initialize the timer and start the game.
    */
   public void startGame() {
      timer = new Timer(time, new ActionListener() {
         @Override
         // every time a timer interrupt appears the snake moves
         public void actionPerformed(ActionEvent e) {
            Console.displayOff();
            if (!snake.move()) {
               gameOver();
            } else {
               Console.setForeground(Color.BLACK);
               Console.setBackground(Color.BLACK);
               Console.clear();
               drawSnake();
               drawApple();
            }
            Console.displayOn();
         }
      });
      // start the timer
      timer.setCoalesce(true);
      timer.start();
   }
   /**
    * Listen to the keyboard and set the new direction.
    *
    * @param c
    *            The new direction.
    */
   public static void listen(char c) {
      snake.setDir(c);
   }
   /**
    * Draw the "Game Over" screen with current score: snake.length - initial
    * length of snake
    */
   private static void gameOver() {
      timer.stop();
      Console.displayOff();
      Console.clear();
      int x = 10;
      int y = 6;
      Console.gotoXY(x, y);
      Console.setBackground(Color.WHITE);
      Console.write("   ");
      x = x + 5;
```

```
        Console.gotoXY(x, y);
        Console.write("  ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("    ");
        x = 9;
        y++;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("  ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write("  ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = 9;
        y++;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("  ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write("    ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("   ");
        x = 9;
        y++;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = 10;
        y++;
        Console.gotoXY(x, y);
        Console.write("   ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("    ");
        y++;
        x = 10;
        y++;
        Console.gotoXY(x, y);
        Console.write("  ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("    ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write("   ");
        x = 9;
        y++;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = 9;
        y++;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write("   ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write("    ");
        x = 9;
        y++;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 2;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = 10;
        y++;
        Console.gotoXY(x, y);
        Console.write("  ");
        x = x + 6;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 4;
        Console.gotoXY(x, y);
        Console.write("    ");
        x = x + 5;
        Console.gotoXY(x, y);
        Console.write(" ");
        x = x + 3;
        Console.gotoXY(x, y);
        Console.write(" ");
        y++;
        y++;
        y++;
```

```
      x = 10;
      Console.gotoXY(x, y);
      Console.setForeground(Color.BLACK);
      Console.write("Your score:" + (snake.getLength() -
ISnake.DEFAULT_START_LENGTH));
      Console.displayOn();
      Console.displayOff();
   }
   /**
    * @return the appleBody
    */
   public Body getAppleBody() {
      return appleBody;
   }
   /**
    * Draw the snake based on values in snake.body (LinkedList).
    */
   private static void drawSnake() {
      Console.setForeground(Color.DARK_GRAY);
      Console.setBackground(Color.GREEN);
      for (int i = 0; i < ISnake.body.size(); i++) {
         int x = ISnake.body.get(i).getPositionX();
         int y = ISnake.body.get(i).getPositionY();
         Console.gotoXY(x, y);
         Console.write("~");
      }
   }
   /**
    * Draw the apple and the instructions.
    */
   private static void drawApple() {
      Console.setForeground(Color.WHITE);
      Console.setBackground(Color.BLACK);
      Console.gotoXY(0, 0);
      Console.write("w: up, a: left, s: down, d: right");
      Console.setBackground(Color.RED);
      Console.setForeground(Color.GREEN);
      Console.gotoXY(appleBody.getPositionX(), appleBody.getPositionY());
      Console.write("O");
      Console.setForeground(Color.WHITE);
      Console.setBackground(Color.BLACK);
   }
   /**
    * Every time the snake eats an apple, a new apple has to be set.
    */
   public void setApple() {
      // apple must fit in the console columns and rows
      double x = -1;
      double y = -1;
      do {
         x = Math.round(Math.random() * Console.DEFAULT_COLS);
         y = Math.round(Math.random() * (Console.DEFAULT_ROWS - 2)) + 1;
         appleBody.setPositionX((int) x);
         appleBody.setPositionY((int) y);
      } while (ISnake.body.contains(appleBody) || (appleBody.getPositionX() ==
Console.DEFAULT_COLS - 1
            && appleBody.getPositionY() == Console.DEFAULT_ROWS - 1));
      drawApple();
   }
}

package snake;
import java.awt.Color;
```

```java
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.KeyEventDispatcher;
import java.awt.KeyboardFocusManager;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import java.util.concurrent.CountDownLatch;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.Timer;
/**
 * Old-School-Konsole-Fenster angelehnt an die CRT-Unit von Borland.
 *
 * @author Denny Schneeweiß und Sven Löffler
 */
public class Console implements KeyListener {

   //
-------------------------------------------------------------------------------
   /** Singleton-Variable */
   private static Console INSTANCE = null;

   //
----------------------------------------------------------------------------
   static final int DEFAULT_COLS = 40;
   static final int DEFAULT_ROWS = 27;
   private static final String DEFAULT_TITLE = "PSCB Console";
   private static final int DEFAULT_COL_WIDTH = 20;
   private static final int DEFAULT_ROW_HEIGHT = 24;
   private static final int PADDING = 10;

   private static final int FONT_UP_SHIFT = 4;
   private static final int FONT_RIGHT_MARGIN = 3;
   //
----------------------------------------------------------------------------

   private int cols      = DEFAULT_COLS;
   private int rows      = DEFAULT_ROWS;
   private int colWidth   = DEFAULT_COL_WIDTH;
   private int rowHeight  = DEFAULT_ROW_HEIGHT;

   private Font font = new Font("Arial", Font.BOLD, 20);

   //
----------------------------------------------------------------------------

   private int bufferedimageWidth;
   private int bufferedimageHeight;
   private BufferedImage frontBuffer;
   private BufferedImage backBuffer;
   private BufferedImage imgBuffer1;
   private BufferedImage imgBuffer2;
   private Graphics backBufferGraphics;
   private JFrame frame;
   private JPanel drawPanel;
   private int cursorX = 0;
   private int cursorY = 0;
   private Color background = Color.BLACK;
   private Color foreground = Color.WHITE;
   private boolean drawInstantly = true;
   private boolean showGrid = false;
   private boolean showCursor = false;
   //
----------------------------------------------------------------------------

   private static Console instance() {

      // prüfen, ob das Singleton schon erzeugt wurde
      if(INSTANCE == null) {

         // Console-Singleton erzeugen
         INSTANCE = new Console(DEFAULT_TITLE, DEFAULT_ROWS, DEFAULT_COLS);
      }

      // Singleton zurückliefern
      return INSTANCE;
   }

   //
----------------------------------------------------------------------------

   private ColorChar[][] inhalt = null;

   private  Console(String titel, int rows, int cols) {

      this.rows = rows;
      this.cols = cols;

      frame = new JFrame(titel);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(this.cols * this.colWidth + 60, this.rows * this.rowHeight +
60);
      frame.addKeyListener(this);

      //
----------------------------------------------------------------------------
      // Zeichenmethode überschreiben, sodass der FrontBuffer genutzt wird.
      // Repaint wird vom Timer angestoßen.
      drawPanel = new JPanel(false) {
         private static final long serialVersionUID = 1L;
         @Override
         protected void paintComponent(Graphics g) {

            super.paintComponent(g);

            if(frontBuffer != null) {

               // Synchronisation auf den FrontBuffer (blockieren, wenn der
gerade geändert wird)
               synchronized (frontBuffer) {

                  if (frontBuffer != null) {
                     g.drawImage(frontBuffer, 0, 0, getWidth(), getHeight(),
null);
                  }
               }
            }

         }

      };

      //
```

```java
----------------------------------------------------------------------------
      // Animationstimer, der Buffer wird alle XX Millisekunden neu dargestellt

      Timer animationTimer = new Timer(30, new ActionListener() {

         @Override
         public void actionPerformed(ActionEvent e) {
            drawPanel.repaint();
         }
      });

      animationTimer.start();
      //
----------------------------------------------------------------------------



      frame.add(drawPanel);
      Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
      frame.setLocation((d.width - frame.getSize().width) / 2, (d.height -
frame.getSize().height) / 2);
      frame.setVisible(true);

      //
----------------------------------------------------------------------------
      // Double-Buffering aufsetzen

      bufferedimageWidth = drawPanel.getWidth();
      bufferedimageHeight = drawPanel.getHeight();

      imgBuffer1 = new BufferedImage(bufferedimageWidth, bufferedimageHeight,
BufferedImage.TYPE_INT_RGB);
      imgBuffer2 = new BufferedImage(bufferedimageWidth, bufferedimageHeight,
BufferedImage.TYPE_INT_RGB);

      frontBuffer = imgBuffer1;
      backBuffer = imgBuffer2;
      backBufferGraphics = backBuffer.getGraphics();

      backBufferGraphics.setColor(background);
      backBufferGraphics.fillRect(0, 0, bufferedimageWidth,
bufferedimageHeight);

      //
----------------------------------------------------------------------------
      // Inhalts-Array initialisieren

      this.inhalt = new ColorChar[this.cols][this.rows];

      for (int x = 0; x < this.cols; x++) {
         for (int y = 0; y < this.rows ; y++) {
            inhalt[x][y] = new ColorChar(' ', this.foreground,this.background);
         }
      }

      backBuffer.getGraphics().setFont(this.font);
      frontBuffer.getGraphics().setFont(this.font);
      frame.repaint();
   }
   /** Setzt den Cursor an die übergebenen Koordinaten innerhalb der Console */
   public static void gotoXY(int x, int y) {

      instance().setCursorXYP(x, y);

      if(instance().drawInstantly) {
         instance().drawArray();
      }
   }

   private void setCursorXYP(int x, int y) {

      if(x >= this.cols) {
         this.cursorX = this.cols-1;
      }
      else if(x < 0) {
         this.cursorX = 0;
      }
      else {
         this.cursorX = x;
      }

      if(y >= this.rows) {
         this.cursorY = this.rows-1;
      }
      else if(y < 0) {
         this.cursorY = 0;
      }
      else {
         this.cursorY = y;
      }
   }
   //
----------------------------------------------------------------------------
   /** Setzt die Hintergrundfarbe. */
   public static void setBackground(Color background) {

      instance().background = background;
   }

   //
----------------------------------------------------------------------------
   /** Setzt die Vordergrundfarbe. */
   public static void setForeground(Color foreground) {
      instance().foreground = foreground;
   }
   //
----------------------------------------------------------------------------
   /** Setzt das übergebene Zeichen an die aktuelle Cursor-Position mit
    *  den momenten gültigen Einstellungen für Vorder- und Hintergrundfarbe.
    */

   public static void setChar(char c) {
      Console instance = instance();

      instance.inhalt[instance.cursorX][instance.cursorY].setCharacter(c);
      instance.inhalt[instance.cursorX]
[instance.cursorY].setForgroundColor(instance.foreground);
      instance.inhalt[instance.cursorX]
[instance.cursorY].setBackgroundColor(instance.background);
      if(instance.drawInstantly) {
         instance.drawArray();
      }
   }

   //
----------------------------------------------------------------------------
   /** Schreibt die übergebene Zeichenkette mit den aktuellen Einstellungen für
Vorder-
    *  und Hintergrundfarbe.
```

```
     */
   public static void write(String s) {

      instance().writeP(s);
   }

   //
----------------------------------------------------------------------------
   /** Schreibt die übergebene Zeichenkette mit den aktuellen Einstellungen für
Vorder-
     *  und Hintergrundfarbe und fügt einen Zeilenumbruch an.
     */
   public static void writeln(String s) {

      instance().writeP(s + '\n');
   }

   //
----------------------------------------------------------------------------
   /** Bewirkt einen Zeilenumbruch. */
   public static void writeln() {

      instance().writeP("\n");
   }

   //
----------------------------------------------------------------------------
   private void writeP(String s) {

      for (int i = 0; i < s.length(); i++) {

         char ch = s.charAt(i);

         if(Character.isWhitespace(ch) && ch != '\n' && ch != ' ') {

            continue;
         }

         if(s.charAt(i) == '\n') {

            cursorX = 0;
            cursorY++;
         }
         else {

            this.inhalt[this.cursorX][this.cursorY].setCharacter(s.charAt(i));
            this.inhalt[this.cursorX]
[this.cursorY].setForgroundColor(this.foreground);
            this.inhalt[this.cursorX]
[this.cursorY].setBackgroundColor(this.background);
            cursorX++;

            if (cursorX >= this.cols) {
               cursorX = 0;
               cursorY++;
            }
         }

         if (cursorY >= this.rows) {

            Color fg = inhalt[this.cols-1][this.rows-1].forgroundColor;
            Color bg = inhalt[this.cols-1][this.rows-1].backgroundColor;

            cursorY = this.rows -1;
            for (int x = 0; x < this.cols; x++) {
               for (int y = 1; y < this.rows; y++) {

                  inhalt[x][y-1].character = inhalt[x][y].character;
                  inhalt[x][y-1].forgroundColor = inhalt[x][y].forgroundColor;
                  inhalt[x][y-1].backgroundColor = inhalt[x][y].backgroundColor;
               }
            }

            // neue Zeile mit den aktuellen Pinseleinstellungen belegen (HG, BG,
Leer)
            for (int x = 0; x < this.cols; x++) {
               inhalt[x][this.rows-1].character = ' ';
               inhalt[x][this.rows-1].forgroundColor = fg; // this.foreground;
               inhalt[x][this.rows-1].backgroundColor = bg; // this.background;
            }
         }
      }
      if(this.drawInstantly) {
         drawArray();
      }
   }

   //
----------------------------------------------------------------------------
   /** Löscht die Konsole und setzt die Einstellunge für Vorder- und
    *  Hintergrundfarbe aller Zeichen auf die aktuellen Einstellungen der
Konsole.
    */
   public static void clear() {

      instance().clearP();
   }

   //
----------------------------------------------------------------------------
   private void clearP() {

      this.setCursorXYP(0, 0);

      for (int x = 0; x < this.cols; x++) {
         for (int y = 0; y < this.rows; y++) {

            inhalt[x][y].character = ' ';
            inhalt[x][y].backgroundColor = this.background;
            inhalt[x][y].forgroundColor = this.foreground;
         }
      }

      if(this.drawInstantly) {
         drawArray();
      }
   }


   //
----------------------------------------------------------------------------
   /** Zeichnet das Array neu in den Backbuffer und macht anschließend den
Buffertausch. */

   private void drawArray() {

      backBufferGraphics.setFont(this.font);
      backBufferGraphics.clearRect(0, 0, bufferedimageWidth,
bufferedimageHeight);

      for (int x = 0; x < this.cols; x++) {
         for (int y = 0; y < this.rows; y++) {

            if (inhalt[x][y] != null) {


               backBufferGraphics.setColor(inhalt[x][y].getBackgroundColor());

               backBufferGraphics.fillRect (
                     PADDING + x * this.colWidth,
                     PADDING + y * this.rowHeight,
                     this.colWidth,
                     this.rowHeight
                  );


               if(this.showGrid) {

                  backBufferGraphics.setColor(Color.GRAY);

                  backBufferGraphics.drawRect (
                        PADDING + x * this.colWidth,
                        PADDING + y * this.rowHeight,
                        this.colWidth,
                        this.rowHeight
                  );
               }

               backBufferGraphics.setColor(inhalt[x][y].getForgroundColor());

               backBufferGraphics.drawString (
                  inhalt[x][y].getCharacter() + "",
                  PADDING + x * this.colWidth + FONT_RIGHT_MARGIN,
                  PADDING + y * this.rowHeight + this.rowHeight - FONT_UP_SHIFT
               );

            }
         }
      }

      if(this.showCursor) {

         backBufferGraphics.setColor(Color.white);
         backBufferGraphics.drawRect(
               PADDING + this.cursorX * this.colWidth,
               PADDING + this.cursorY * this.rowHeight,
               this.colWidth,
               this.rowHeight
            );
      }


      backBufferGraphics.setColor(Color.WHITE);
      backBufferGraphics.drawRect(
            PADDING -1,
            PADDING -1,
            this.cols * this.colWidth + 1,
            this.rows * this.rowHeight + 1
         );
      // frame.repaint();
      switchBuffer();
   }

   //
-----------------------------------------------------------------------------
   /** Lässt das Programm für die angegebene Zeit warten. */
   public static void wait(int time) {

      try {
         Thread.sleep(time);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
   //
-----------------------------------------------------------------------------
   /** Sorgt dafür, dass die folgenden Konsolenaktionen nicht dargestellt
werden,
    *  bis displayOn() aufgerufen wird. Dient zur Beschleunigung und wirkt sich
nur
    *  auf Schreib- und Zeichenmethoden aus.
    */
   public static void displayOff() {

      instance().drawArray();
      instance().drawInstantly = false;
   }

   /** Aktualisiert die Anzeige der Konsole, sofern vorher displayOff()
    *  aufgerufen wurde.
    */
   public static void displayOn() {

      instance().drawInstantly = true;
      instance().drawArray();
   }
   //
-----------------------------------------------------------------------------
   /** Unterbricht das Programm so lange, bis die Leertaste gedrückt wurde. */
   public static void waitSpaceKey() {

      instance().waitKeyP();
   }
   //
-----------------------------------------------------------------------------
   private void waitKeyP() {
       final CountDownLatch latch = new CountDownLatch(2);
       KeyEventDispatcher dispatcher = new KeyEventDispatcher() {

           // Anonymous class invoked from EDT
           public boolean dispatchKeyEvent(KeyEvent e) {
               if (e.getKeyCode() == KeyEvent.VK_SPACE)
                   latch.countDown();
               return false;
           }
       };

KeyboardFocusManager.getCurrentKeyboardFocusManager().addKeyEventDispatcher(disp
atcher);
       try {
         latch.await();
      } catch (InterruptedException e1) {
         e1.printStackTrace();
      }
       // current thread waits here until countDown() is called

KeyboardFocusManager.getCurrentKeyboardFocusManager().removeKeyEventDispatcher(d
ispatcher);


       try {
         Thread.sleep(50);
      } catch (InterruptedException e1) {
         e1.printStackTrace();
      }
   }

   //
----------------------------------------------------------------------------
   /** Wartet auf einen Tastendruck und gibt den Zeichencode als char-Wert
zurück. */
   public static char readKey() {

      return instance().readKeyP();
   }

   //
----------------------------------------------------------------------------
   private class KeyChar {

      private char ch;
      public void setChar(char c) {
         this.ch = c;
      }
   }

   private char readKeyP() {

       final CountDownLatch latch = new CountDownLatch(2);
       final KeyChar kc = new KeyChar();

       KeyEventDispatcher dispatcher = new KeyEventDispatcher() {
           // Anonymous class invoked from EDT
           public boolean dispatchKeyEvent(KeyEvent e) {
               latch.countDown();
               kc.setChar(e.getKeyChar());
               return false;
           }
       };


KeyboardFocusManager.getCurrentKeyboardFocusManager().addKeyEventDispatcher(disp
atcher);
       try {
         latch.await();
      } catch (InterruptedException e1) {
         e1.printStackTrace();
      }
       // current thread waits here until countDown() is called

KeyboardFocusManager.getCurrentKeyboardFocusManager().removeKeyEventDispatcher(d
ispatcher);


       try {
         Thread.sleep(50);
      } catch (InterruptedException e1) {
         e1.printStackTrace();
      }

     return kc.ch;
   }

   //
-----------------------------------------------------------------------------
   /** Schaltet die Darstellung des Zeilen- und Spaltenrasters ein und aus. */

   public static void showGrid(boolean show) {

      instance().showGrid = show;

      if(INSTANCE.drawInstantly)
         instance().drawArray();
   }

   //
-----------------------------------------------------------------------------
   /** Schaltet die Darstellung des Cursors ein und aus. */

   public static void showCursor(boolean show) {

      instance().showCursor = show;

      if(INSTANCE.drawInstantly)
         instance().drawArray();
   }
   //
-----------------------------------------------------------------------------
   private class ColorChar {
      private Color backgroundColor = Color.BLUE;
      private Color forgroundColor = Color.YELLOW;
      private char character;
      public ColorChar(char character, Color fgColor, Color bgColor) {
         this.character = character;
         this.backgroundColor = bgColor;
         this.forgroundColor = fgColor;
      }
      public Color getBackgroundColor() {
         return backgroundColor;
      }
      public void setBackgroundColor(Color backgroundColor) {
         this.backgroundColor = backgroundColor;
      }
      public Color getForgroundColor() {
         return forgroundColor;
      }
      public void setForgroundColor(Color forgroundColor) {
         this.forgroundColor = forgroundColor;
      }
      public char getCharacter() {
         return character;
      }
      public void setCharacter(char character) {
         this.character = character;
      }
   }
   //
-----------------------------------------------------------------------------
   /** Tauscht Back- und Frontbuffer. */
   private void switchBuffer() {

      synchronized (frontBuffer) {
         if(backBuffer == imgBuffer1) {

            backBuffer = imgBuffer2;
            frontBuffer = imgBuffer1;
         }
         else {
            backBuffer = imgBuffer1;
            frontBuffer = imgBuffer2;
         }

         backBufferGraphics = backBuffer.getGraphics();

      }

   }
   @Override
   public void keyTyped(KeyEvent e) {
      // TODO Auto-generated method stub
      char c = e.getKeyChar();
      GameBoard.listen(c);

   }
   @Override
   public void keyPressed(KeyEvent e) {
      // TODO Auto-generated method stub
      char c = e.getKeyChar();
      GameBoard.listen(c);
   }
   @Override
   public void keyReleased(KeyEvent e) {
      // TODO Auto-generated method stub

   }

}

//@author maximilian raspe
package snake;
public class Body {
    private int x;
    private int y;
    //constructor der x und y übergeben bekommt und dann x,y setzt
    public Body(int x, int y) {
        this.x = x;
        this.y = y;
    }
    //getter und setter methoden für x und y
    public int getPositionX() {
        return x;
    }
    public int getPositionY() {
        return y;
    }
    public void setPositionX(int x) {
        this.x = x;
    }
    public void setPositionY(int y) {
       this.y = y;
    }
    @Override
    public String toString() {
        return "(" + this.x + "," + this.y + ")";
    }
    @Override
    public boolean equals(Object obj) {
        if(!(obj instanceof Body)) {
            return false;
        }
        Body m = (Body) obj;
        if(this.x != m.x) {
            return false;
        }
        if(this.y != m.y) {
            return false;
        }
       return true;
    }
}
```

```
//@author maximilian raspe
public class Loewe extends Tier {
   public Loewe(String tiername) {
      name = tiername;
      art = "Löwe";
      Schreiverhalten bruellen = new Bruellen();
      schreiverhalten = bruellen;
   }
}
//@author maximilian raspe
public interface Schreiverhalten {
   public void schreien();
}
//@author maximilian raspe
public abstract class Tier implements Schreiverhalten {

   protected Schreiverhalten schreiverhalten;
   protected String name;
   protected String art;

   public String getArt() {
      return art;
   }

   public String getName() {
      return name;
   }

   public Schreiverhalten getSchreiverhalten() {
      return schreiverhalten;
   }
        //führt schrei aus
   public void schreien() {
      schreiverhalten.schreien();
   }
        //gibt namen und art des tieres zurück
```

```
    public String toString() {
        return this.getName() + " der " + this.getArt();
    }
}
//@author maximilian raspe
public class Troeten implements Schreiverhalten {
    @Override
    public void schreien() {
        System.out.println("TÖRÖÖÖ!");
    }
}
//@author maximilian raspe
public class Wolf extends Tier {

    public Wolf(String tiername) {
        name = tiername;
        art = "Wolf";
        Schreiverhalten heulen = new Heulen();
        schreiverhalten = heulen;
    }
}
//@author maximilian raspe
public class Zoo {
    public static void main(String[] args) {
        Tier[] viechaz = new Tier[4];     //erstellt array mit tieren
                    //füllt array mit neuen tieren und führt diesen aus
        viechaz[0] = new Elefant("Justus");
        System.out.println(viechaz[0]);
        viechaz[0].schreien();
        if (viechaz[0].getArt() == "Elefant") new Elefant().stampfen();
        System.out.println();
        viechaz[1] = new AfrikanischerElefant("Justyn");
        System.out.println(viechaz[1]);
        viechaz[1].schreien();
        if(viechaz[1].getArt() == "Afrikanischer Elefant") {
            new Elefant().stampfen();
        }
        System.out.println();
        viechaz[2] = new Wolf("Erik");
        System.out.println(viechaz[2]);
        viechaz[2].schreien();
        System.out.println();
        viechaz[3] = new Loewe("Lobo");
        System.out.println(viechaz[3]);
        viechaz[3].schreien();
        if (viechaz[3].getArt() == "Löwe") System.out.println("Er ist der König der Tiere");


    }

}
//@author maximilian raspe
```

```
}
public class AfrikanischerElefant extends Elefant {
        //konstruktor für den afrikanischen elefanten
    public AfrikanischerElefant(String tiername) {
        super(tiername);
        super.art = "Afrikanische Elefant";
    }
        //default konstruktor
    public AfrikanischerElefant() {
    }
    public String zeigeMarkenzeichen() {
        return name + " wackelt mit seinen großen Ohren";
    }
        //überschreibt die toString methode aus tier
    public String toString() {
        return super.toString()+ "\n" + this.zeigeMarkenzeichen();
    }
}
//@author maximilian raspe
public class Bruellen implements Schreiverhalten {
    @Override
    public void schreien() {
        System.out.println("ROOAR!");
    }
}
//@author maximilian raspe
public class Elefant extends Tier {
        //constructor für den elefanten
    public Elefant(String tiername) {
        name = tiername;
        art = "Elefant";
        Schreiverhalten troeten = new Troeten();
        schreiverhalten = troeten;
    }
    public Elefant() {
    }
    public void stampfen() {
        System.out.println("Wenn er auf den Boden stampft zittert die Erde");
    }

}
//@author maximilian raspe
public class Heulen implements Schreiverhalten{
    @Override
    public void schreien() {
        System.out.println("AUUUUU");
    }
}
```