Prof. Dr.-Ing. Elmar Nöth und Priv.-Doz. Dr.-Ing. Peter Wilke
Lehrstuhl für Mustererkennung
Universität Erlangen-Nürnberg

12. Oktober 2011

Schriftliche Prüfung zu der Vorlesung Grundlagen der Informatik Oktober 2011

(Prüfungsnummern 11 104 112, 11 177 1100, 30601)

Angaben zur Person:

Name, Vorname .....

Geburtsdatum .....

Matrikelnummer .....

Studiengang .....

Nicht von der Kandidatin/dem Kandidaten auszufüllen!

Bewertung:

Aufgabe	1	2	3	4	5	6	7	$\sum$	Note
Punktezahl									

Organisatorische Hinweise

Die folgenden Hinweise bitte aufmerksam lesen und die Kenntnisnahme durch Unterschrift bestätigen!

- Bitte legen Sie Ihren Studentenausweis und einen Lichtbildausweis zur Personenkontrolle bereit!
- Schreiben Sie deutlich und ausschließlich mit blauer oder schwarzer Tinte. Benutzen Sie keinen Bleistift. Unleserliche Antworten gehen nicht in die Bewertung ein.
- Hilfsmittel (außer Schreibmaterial) sind nicht zugelassen. Dies gilt auch für Taschenrechner, Mobiltelefone, elektronische Assistenten, etc.
- Fragen zu den Prüfungsaufgaben werden grundsätzlich nicht beantwortet!
- **Überprüfen Sie die Prüfungsaufgaben auf Vollständigkeit (19 Seiten inklusive Deckblatt) und einwandfreies Druckbild!** Vergessen Sie nicht, auf dem Deckblatt die Angaben zur Person einzutragen!
- Die Bearbeitungszeit beträgt 90 Minuten. Da Sie maximal 90 Punkte erreichen können, ist aus den möglichen Punkten pro Aufgabe leicht die Zeit zu berechnen, die Sie zum Lösen der jeweiligen Aufgabe investieren sollten. Die angegebene Punkteverteilung gilt unter Vorbehalt.
- Auf Ihrem Platz befindet sich 1 Blatt Schmierpapier. **Das Schmierpapier darf nicht mit abgegeben werden.** Die Lösung einer Aufgabe muss auf das Aufgabenblatt geschrieben werden, und zwar in den freien Raum, der jeweils der Aufgabe folgt. Sollte der Platz nicht ausreichen, so müssen Sie bei der Aufsicht zusätzliche Formblätter anfordern und einheften lassen.
- Wenn Sie die Prüfung aus gesundheitlichen Gründen abbrechen müssen, so muss Ihre Prüfungsunfähigkeit durch eine Untersuchung in der Universitätsklinik nachgewiesen werden. Melden Sie sich in jedem Fall bei der Aufsicht und lassen Sie sich das entsprechende Formular aushändigen.

Erklärung

Durch meine Unterschrift bestätige ich den Empfang der vollständigen Klausurunterlagen und die Kenntnisnahme der obigen Informationen.

Erlangen, den 12. Oktober 2011 .....
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis unter Angabe der Matrikelnummer anonymisiert veröffentlicht wird:

ja: ☐ nein: ☐

Erlangen, den 12. Oktober 2011 .....
(Unterschrift)

Aufgabe 1 (10 Punkte)

Kreuzen Sie die richtige Antwort auf die jeweilige Frage an. Pro Frage ist immer nur eine Antwort richtig.

1. Welche Aussage trifft auf die Komplementbildung bei der Zahlendarstellung zu?
   (a) Zu einer Zahl sind deren Einer- und Zweier-Komplement immer verschieden. ☐
   (b) Für die Darstellung eines negativen Exponenten wird keine Komplementbildung benutzt. ☐
   (c) Die Null hat kein Vorzeichen und wird deshalb im Einer- und Zweier-Komplement immer gleich dargestellt. ☐

2. Welche Aussage trifft für Konstanten in Java zu?
   (a) Einer Konstanten kann vom Programmierer genau 0-mal ein Wert zugeordnet werden. ☐
   (b) Einer Konstanten kann vom Programmierer genau 1-mal ein Wert zugeordnet werden. ☐
   (c) Einer Konstanten kann vom Programmierer höchstens 1-mal ein Wert zugeordnet werden. ☐

3. Seien `x` und `y` Variablen vom Typ `int`. Der Ausdruck
   (a) „`x = y`“ ist dann der Test auf Gleichheit. ☐
   (b) „`x == y`“ ist dann der Test auf Wert-Gleichheit. ☐
   (c) „`x === y`“ ist dann der Test auf Struktur-Gleichheit. ☐

4. Für Zahlen gilt die Aussage:
   (a) Die Stellenzahl der Mantisse einer Gleitkommazahl hat einen Einfluss auf die Genauigkeit. ☐
   (b) Eine Gleitkommazahl besteht aus Vorzeichen, Signatur und Mantisse. ☐
   (c) Zwischen zwei beliebigen ganzen Zahlen liegen unendlich viele weitere ganze Zahlen. ☐

5. Für `if`-Anweisungen gilt in Java:
   (a) Der `default`-Zweig ist zwingend erforderlich. ☐
   (b) Im JA-Zweig sind nur Block-Anweisungen erlaubt, nicht jedoch einfache Anweisungen. ☐
   (c) Ein vorhandener `else`-Teil wird immer dem letzten sichtbaren `if` zugeordnet, das noch kein `else` besitzt. ☐

6. Welche Aussage trifft auf binäre Sortierbäume zu?

   (a) Alle Knoten einschließlich der Wurzel haben immer genau 1 Vorgänger und höchstens 2 Nachfolger. ☐

   (b) Alle Knoten einschließlich der Wurzel haben immer entweder 1 oder 2 Nachfolger. ☐

   (c) Alle Knoten einschließlich der Wurzel müssen keinen Nachfolger haben. ☐

7. Welche Aussage trifft auf Modifikatoren von Attributen zu?

   (a) Der Modifikator `void` kann nur zur Spezifikation von Klassen und Methoden verwendet werden. ☐

   (b) Der Modifikator `protected` kann nie gleichzeitig mit dem Modifikator `final` verwendet werden. ☐

   (c) Der Modifikator `public` erlaubt den am wenigsten eingeschränkten Zugriff der drei Modifikatoren `public`, `private` und `protected`. ☐

8. Welche Aussage gilt in der objektorientierten Programmierung für den Mechanismus der Vererbung?

   (a) Methoden in abgeleiteten Klassen überladen nur abstrakte Methoden gleicher Signatur in der Basisklasse. ☐

   (b) Gibt es in der Basisklasse den parameterlosen Standardkonstruktor nicht, muss im Konstruktor der abgeleiteten Klasse explizit ein Konstruktor der Basisklasse aufgerufen werden. ☐

   (c) Ein Objekt einer abgeleiteten Klasse kann in Java nicht einer Variablen der Basisklasse zugewiesen werden. ☐

9. Welche Aussage trifft auf Suchalgorithmen zu?

   (a) Divide-and-Conquer Algorithmen können nur rekursiv implementiert werden. ☐

   (b) Greedy-Algorithmen basieren immer auf einem sequentiellen Suchansatz. ☐

   (c) Für einen Suchalgorithmus ist ein exponentieller Aufwand in der Regel im praktischen Einsatz nicht akzeptabel. ☐

10. Welche Aussage trifft auf Sortierverfahren zu?

    (a) Für einen Sortieralgorithmus ist ein exponentieller Aufwand in der Regel im praktischen Einsatz nicht akzeptabel. ☐

    (b) Sortierverfahren sind immer kaskadenartig rekursiv oder haben einen exponentiellen Aufwand. ☐

    (c) Quicksort zeichnet sich gegenüber allen anderen Sortieralgorithmen dadurch aus, dass es keine Ausgangssituation gibt, in der Algorithmus nicht zu einem Ende kommt. ☐

☐

Aufgabe 2 (17 Punkte)

1. Das folgende Programm soll dazu dienen, zu überprüfen, ob ein Punkt mit den Koordinaten `(x,y)` in einem Rechteck liegt. Das Rechteck ist durch seine Position (`x` und `y`) sowie seine Größe (`breite` und `hoehe`) definiert. Die Klasse Rechteck besitzt dazu die Methode `enthaelt`. Sie soll `true` zurückgeben, wenn der Punkt innerhalb liegt, und `false` sonst.

   Im Hauptprogramm werden zuerst ein Rechteck angelegt und die Koordinaten eines Punktes definiert. Danach wird die Methode `enthaelt` benutzt, um zu testen, ob der Punkt im Rechteck liegt. Das Ergebnis wird auf `stdout` ausgegeben. Die Ausgabe des Programms lautet somit:

   ```
   Der Punkt (10, 10) liegt außerhalb.
   ```

   Verbessern Sie die sieben im Programm enthaltenen semantischen und syntaktischen Fehler, so dass es sich danach um ein fehlerfreies Java-Programm handelt, welches die gestellte Aufgabe erfüllt. Schreiben Sie das Programm nicht neu und korrigieren Sie nicht mehr als sieben Fehler!

```
public class Rechteck {
    /** Position und Größe des Rechtecks */
    protected int x = 0, y = 0.5, breite = 0, hoehe = 0;

    /** Konstruktor */
    Rechteck(int x, int y, int breite, int hoehe) {
        this.x = x; this.y = y; this.breite = breite; this.hoehe = hoehe;
    }
    /** Prüft, ob ein Punkt im Rechteck enthalten ist.
     * @param {int} x, y - Koordinaten des Punktes */
    public boolean enthaelt(int x, float y) {
        return ! ( (x < this.x) &&
                   (x >= this.x() + this.breite) ||
                   (y < this.y) ||
                   (y >= this.x + this.hoehe) );
    }

    public static void main(String[] args) {
        Rechteck r = Rechteck(0, 0, 10, 10);
        int punktX = 10, punktY = 10;
        boolean istDrin = r.enthaelt(punktX, punktY);
        System.out.print("Der Punkt (" + punktX + "," + punktY + ") liegt");
        if (istDrin) {
            System.out.println(" im Rechteck.");
        } else {
            System.out.println(" außerhalb.")
        }
    }
}
```

2. Geben Sie präzise an, was beim Ablauf des auf der nächsten Seite stehenden Programms der Reihe nach auf `stdout` ausgegeben wird.

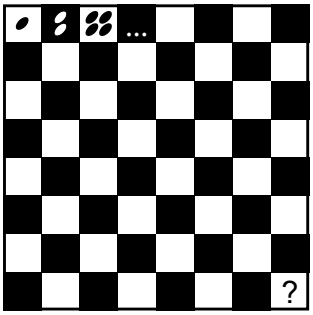
   Auf `stdout` wird ausgegeben:

```
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
```

```
public class BuchstabenSchubser {
    static int x = 0;
    int y, z;

    private int ausgeben() {
        System.out.println("x: " + x);
        return ++z;
    }

    private static void feldAusgeben(char[] f, char c) {
        System.out.print(c + ":");
        for (int i = 0; i < f.length; i++) {
            System.out.print(" " + f[i]);
        }
        System.out.print('\n');
    }

    private static void schubse(char[] c, int x, int y) {
        feldAusgeben(c, 'x');
        for (int i = 0; i < c.length - 1; i++) {
            char temp = c[i];
            c[i] = c[i+1];
            c[i+1] = temp;
        }
        feldAusgeben(c, 'y');
    }

    public static void main(String[] args) {
        int y = 3;
        x = 42;
        BuchstabenSchubser sA = new BuchstabenSchubser();
        BuchstabenSchubser sB = new BuchstabenSchubser();
        sA.z = y;
        BuchstabenSchubser.x = 3;
        char[] feld = {'C', 'G', 'D', 'I', 'A', 'B'};

        System.out.println("Ausgabe: " + sA.ausgeben());
        if (sA.x == sB.x) {
            System.out.println("x ist gleich");
        } else {
            System.out.println("x ist unterschiedlich");
        }

        System.out.println("feld[" + x + "] = " + feld[x]);
        schubse(feld, 2, feld.length - 1);
    }
}
```

Aufgabe 3 – Die Weizenkornlegende (12 Punkte)

Sissa ibn Dahir lebte angeblich im dritten oder vierten Jahrhundert n. Chr. in Indien. Nach der Legende gewährte der König ihm einen freien Wunsch. Er wünschte sich Weizenkörner: Auf das erste Feld eines Schachbretts wollte er ein Korn, auf das zweite Feld die doppelte Menge, also zwei, auf das dritte wiederum doppelt so viele, also vier, und so weiter. Der König lachte und war gleichzeitig erbost ob der vermeintlichen Bescheidenheit von Sissa. Als sich der König einige Tage später erkundigte, ob Sissa seine Belohnung in Empfang genommen habe, musste er hören, dass die Rechenmeister die Menge der Weizenkörner noch nicht berechnet hatten!



Mit Hilfe von Computern lässt sich die Zahl inzwischen sehr viel schneller ausrechnen. Die Methode `double koerner(int n)` soll zunächst berechnen, wie viele Reiskörner auf dem `n`-ten Feld liegen. Dazu soll sie folgende Rekursionsformel verwenden:

$$koerner(n) = \begin{cases} 1 & \text{falls } n \leq 1 \\ koerner(n-1) * 2 & \text{sonst} \end{cases} \quad (1)$$

3.1 Geben Sie eine rekursive und eine iterative Implementierung der Methode `koerner` in Java an. Verwenden Sie dazu keine Methoden aus `java.lang.Math`. Rufen Sie in der `main`-Methode der Klasse beide Varianten mit einer Zahl Ihrer Wahl auf und lassen Sie das Ergebnis auf `stdout` ausgeben.
Hinweis: Der Rückgabetyp der Methoden ist `double`, da die Zahlen sehr groß werden. Deshalb sollen Sie auch mit `double`-Variablen rechnen.

```
public class Weizenkoerner {
    // Rekursive Variante der Methode
    static double koerner_rek(int n) {

        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
        ----------------------------------------------------------
    }
```

```
    // Iterative Variante der Methode
    static double koerner_iter(int n) {

        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
    }

    // main Methode
    public static void main(String[] args) {

        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
    }

} // class Weizenkoerner
```

3.2 a) Geben Sie eine Formel an, mit der sich $koerner(n)$ direkt berechnen lässt.

3.2 b) Warum ist der Rückgabetyp der Methoden nicht `int`?

3.2 c) Die Kantenlänge eines Schachbretts beträgt 8 Felder. Wie viele Körner liegen insgesamt darauf? Geben Sie auch hier eine Formel an (und nicht die Zahl).

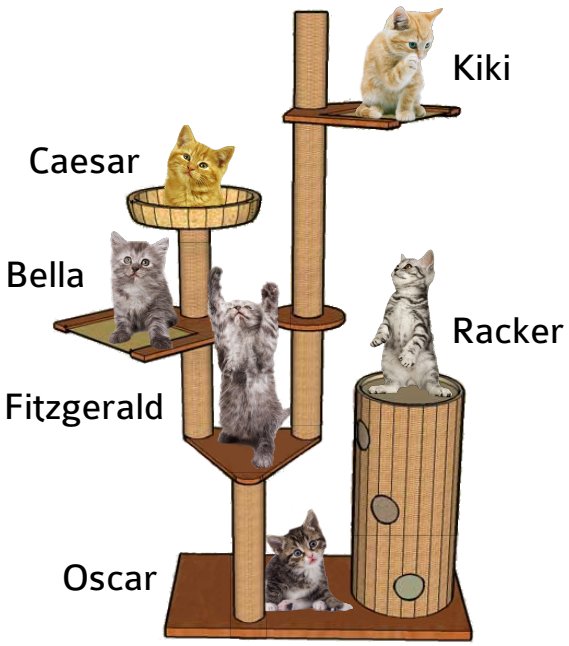
Aufgabe 4 (16 Punkte)

Möchten Sie in einem Programm Daten unsortiert ansammeln, so eignet sich dazu sehr gut die Datenstruktur der verketteten Liste. Sollen einzelne Einträge anhand des Namens schnell gefunden werden, so lässt sich das aber effizienter mit einem binären Suchbaum erledigen.

Gegeben sind die folgenden Klassen, die jeweils ein Listenelement und einen Baumknoten darstellen:

```
public class ListenElement {
    String name;
    ListenElement nachfolger;
}
```

```
public class BaumKnoten {
    String name;
    BaumKnoten links;
    BaumKnoten rechts;
}
```



Beispiel eines binären Katzensuchbaums

Schreiben Sie eine Methode `baum`, die als Parameter eine verkettete Liste übergeben bekommt und einen binären Suchbaum zurückliefert, der alle Namen aus der verketteten Liste enthält. Benutzen Sie dabei die Hilfsmethode `einfuegen`, die einen Namen sortiert in einen bestehenden binären Suchbaum einfügt. Auch diese Methode sollen Sie implementieren.

Für den binären Suchbaum gilt, dass an jedem Knoten der Name eines linken Nachfolgers lexikographisch kleiner sein muss als der Name des Knoten selbst. Entsprechend muss der Name eines rechten Nachfolgers lexikographisch größer sein. Sie können annehmen, dass in der Liste keine Namen mehrfach vorhanden sind. Der Baum darf unbalanciert wachsen.

Hinweise:
Sie dürfen weitere Hilfsmethoden implementieren und verwenden.
Die Methode `int java.lang.String.compareTo(String anotherString)` *liefert eine negative Zahl zurück, wenn das String-Objekt lexikographisch kleiner als* `anotherString` *ist.*

```java
/**
 * Fügt einen Namen in bestehenden binären Suchbaum ein.
 *
 * @param wurzel Wurzelknoten des Baums
 * @param name Einzufügender Name
 */
public static void einfuegen(BaumKnoten wurzel, String name) {

    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
}
```

```
/**
 * Erzeugt binären Suchbaum aus verketteter Liste.
 *
 * @param kopf Erstes Element der Liste
 * @return Wurzelknoten des Baums
 */
public static BaumKnoten baum(ListenElement kopf) {

    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
    -------------------------------------------------------------
}
```

Aufgabe 5 (9 Punkte)

1. Gegeben sei folgende Grammatik $G_1 = (N, T, P, S)$
   mit $P = \{(S \to ABA), (A \to aAaa), (A \to a), (B \to bBb), (B \to bb)\}$.

   - Bestimmen Sie die Mengen N und T.

   - Bestimmen Sie die erzeugte Wortmenge $L(G_1)$.

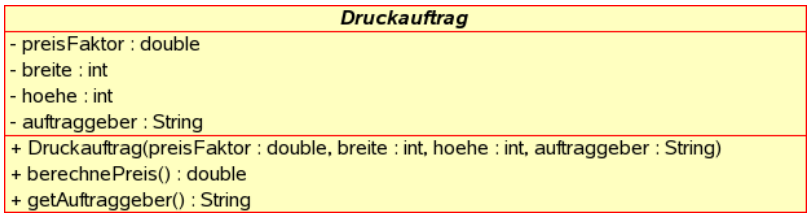
2. Gegeben sei die von der Grammatik G_2 erzeugte Wortmenge
   $L(G_2) = \{a^n b^{n+m} a^m | n, m > 0\}$.
   Geben Sie entweder die Produktionen einer kontextfreien Grammatik an, die diese Wortmenge erzeugt, oder begründen Sie, warum es eine solche kontextfreie Grammatik nicht geben kann.

Aufgabe 6 (26 Punkte)

In einer Druckerei werden Druckaufträge verwaltet. Ihre Aufgabe besteht darin, die Druckaufträge und deren Verwaltung zu implementieren.

1.1 Zunächst soll eine abstrakte Klasse `Druckauftrag` modelliert werden. Bei der Erstellung eines Druckauftrages wird dessen Breite und Höhe festgelegt, sowie ein Preisfaktor, der sich auf die Verwendung eines bestimmten Druckmaterials bezieht. Der Druckauftrag enthält weiterhin ein Attribut `auftraggeber` vom Typ `String`. Dem Konstruktor wird der Auftraggeber ebenfalls als Parameter übergeben.

Eine genauere Beschreibung der einzelnen Methoden finden Sie jeweils vor der Methodendeklaration. Nutzen Sie die Informationen aus dem UML-Diagramm, um die Lücken in dem folgenden Codefragment zu schließen.




```
// Klasse Druckauftrag
-------------------------------------------------------------------

        //Attributdeklarationen
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------

        // Konstruktor
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
        -------------------------------------------------------------
```

```
        /** berechnePreis:
          * Berechnet den Preis des Druckauftrages aus dessen
          * Flaeche, multipliziert mit dem Preisfaktor.          */
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------

        /** getAuftraggeber:
          * Liefert den Auftraggeber zurueck                      */
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
// Ende Klasse Druckauftrag
------------------------------------------------------------
```

1.2 Das Bedrucken verschiedener Materialien wird durch Spezialisierungen des Druckauftrages realisiert. Ein `Papierdruck` ist ein `Druckauftrag` mit einem festen `preisFaktor` von `1.0`. Des Weiteren besitzen alle Objekte vom Typ `Papierdruck` eine fortlaufende Nummer als Attribut, welche von außen zugreifbar ist, aber nach Erstellung des Papierdrucks nicht mehr verändert werden kann.

```
// Klasse Papierdruck
------------------------------------------------------------
        //Attributdeklarationen
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------

        // Konstruktor
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------
        ------------------------------------------------------------

// Ende Klasse Papierdruck
------------------------------------------------------------
```

2. Nun wird die Verwaltung der Druckaufträge in einer `Auftragsschlange` implementiert. Die Druckaufträge sollen als Array mit dem Namen `schlange` gespeichert werden. Die Einträge werden mit `null` initialisiert. Auf das Feld darf nur innerhalb der `Auftragsschlange` und in abgeleiteten Klassen zugegriffen werden.

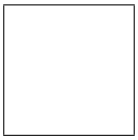
   Modellieren Sie die folgende Funktionalität:

   - Ein Konstruktor soll bereitgestellt werden, der die Anzahl der Einträge im Feld `schlange` als Parameter übergeben bekommt. Das Feld `schlange` wird mit der entsprechenden Anzahl an Einträgen angelegt.
   - Ein parameterloser Konstruktor (Default-Konstruktor) soll bereitgestellt werden, der die Größe des Feldes `schlange` auf 50 Einträge festsetzt.
   - `void neuerAuftrag(Druckauftrag auftrag)`: Das Objekt `auftrag` wird in das Feld `schlange` gespeichert. Die Suche nach einem freien Feldeintrag wird bei dem Feld des zuletzt gespeicherten Auftrages begonnen. Sollten alle Einträge bis ans Feldende belegt sein, wird mit der Suche am Anfang des Feldes wieder begonnen. Sollten alle Felder der `schlange` belegt sein, wird eine `SchlangeVollException` geworfen.
   - `double berechneGesamtpreis(String auftraggeber)`: Liefert den Gesamtpreis aller in der Auftragsschlange gespeicherten Druckaufträge mit Auftraggeber `auftraggeber` zurück.

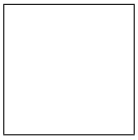
   Zum Bereitstellen der Funktionalität der Klasse werden eventuell weitere Attribute benötigt. Wählen Sie für diese geeignete Datentypen. Die Benennung bleibt Ihnen überlassen. Die Sichtbarkeit aller Attribute soll, sofern nicht anders gefordert, so restriktiv wie möglich definiert werden.

   Hinweis: Verwenden Sie für Ihren Programmcode die hier und auf der folgenden Seite definierte Vorlage! Die Klasse `SchlangeVollException` muss nicht implementiert werden!

```
// Klasse Auftragsschlange
-----------------------------------------------------------------
```



```
    //Attributdeklarationen
    -----------------------------------------------------------
    -----------------------------------------------------------
    -----------------------------------------------------------

    // Konstruktoren
    -----------------------------------------------------------
    -----------------------------------------------------------
    -----------------------------------------------------------
    -----------------------------------------------------------
    -----------------------------------------------------------
    -----------------------------------------------------------
    -----------------------------------------------------------
```



```
        /** neuerAuftrag:
          * Fuegt einen neuen Auftrag in die Schlange ein          */
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------

        /** berechneGesamtpreis:
          * Berechnet den Gesamtpreis aller Auftraege
          * eines Auftraggebers                                     */
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------
        ---------------------------------------------------------------

// Ende Klasse Auftragsschlange
-------------------------------------------------------------
```

3. Eine `Druckerei` besitzt als Attribut eine `Auftragsschlange`, auf welches von außen und von abgeleiteten Klassen nicht zugegriffen werden kann, sowie einen Default-Konstruktor. Zeichnen Sie ein UML-Diagramm, welches die in dieser Aufgabe erstellten Klassen sowie die Beziehungen zwischen den Klassen `Druckauftrag`, `Papierdruck`, `Auftragsschlange` und `Druckerei` darstellt.

   Hinweise: Die Klasse `Druckerei` muss **nicht** implementiert werden.
   Die Klasse `SchlangeVollException` muss **nicht** im Diagramm enthalten sein. Die Klasse `Druckauftrag` ist bereits als UML-Diagramm dargestellt. Ergänzen Sie diese Darstellung auf dieser Seite mit den weiteren Klassen und Beziehungen. Erstellen sie das Diagramm mit allen Methoden einschließlich der Konstruktoren.