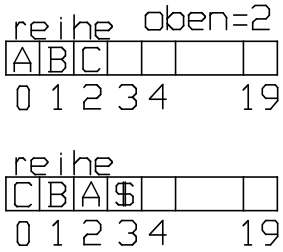
Do., 14.3.2013 Informatik Q1 M (Kr)

1. Klausur 12/II (Q1.2) (A)

Dauer: 3 Zeitstunden (9:15 bis 12:15 Uhr) Name: ___www.r-krell.de___

Hilfsmittel: --

* *Achte auf sorgfältige Darstellung mit vollständigem, nachvollziehbarem Lösungsweg!* *
* *Kommentiere deine Programme!* *



1 Buchstaben-Keller mit Reihung

Sind in einem Keller nie mehr als 19 oder 20 Buchstaben gleichzeitig, so liegt die Implementierung mit einer Reihung (Index 0 bis 19) nahe. Wurde zuerst 'A', danach 'B' und zuletzt 'C' eingekellert, sind die beiden abgebildeten Füllungen möglich (wobei das '$'-Zeichen in Version II automatisch anfangs links im neuen leeren Keller stehen soll, also nicht mit *rein* eingekellert wurde!)

a) Im Unterricht haben wir die Version I (oberes Bild) benutzt. Schreibe nur die bereits angefangene Methode *rein* vollständig in dein Heft.

b) Schreibe nun für die Version II (wo der oberste/letzte Buchstabe immer in *reihe[0]* steht und es kein *oben* gibt) die ganze *public class CharKellerII* komplett mit der *reihe* und Konstruktor (der auch fürs '$' sorgen muss) und allen vier Methoden *istLeer, rein, raus* und *zeige1*! Beachte die Richtung der *for*-Schleife(n)!

c) Nenne Vor- und Nachteile der Versionen I und II und entscheide begründet, welche Version die bessere ist.

```
public class CharKellerI  // für 1a)
{
  char [ ] reihe = new char[20];
  int oben = -1;

  public char zeige1( )
  {  return ( reihe[oben] ); }

  public void rein (char neuerBst)
  { ...
```

2 Buchstaben-Schlange mit Knoten

Eine Schlange für Buchstaben soll dynamisch mit *CharKnoten* realisiert werden.

a) Schreibe den vollständigen Konstruktor *CharKnoten* für den Knoten (s. Kasten)

b) Schreibe den Anfang der Klasse *CharSchlange* mit den beiden Attributen *kopf* und *schwanz* und nur mit der Methode *rein*, die einen neuen Buchstaben ans Ende der (bereits beliebig gefüllten oder vielleicht noch leeren) Schlange anhängt (mit Kommentar!).

c) Erläutere, wieso eine Implementierung mit Knoten als dynamisch gilt, während der Gebrauch einer Reihung wie in Aufgabe 1 als statisch bezeichnet wird. Beschrebe den Unterschied und begründe, ob dynamisch oder statisch besser ist!

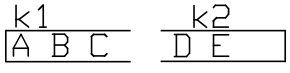
```
public class CharKnoten
{
  char inhalt;
  CharKnoten zeiger;

  public CharKnoten (char x, CharKnoten y)
  { ...
```

3 (einfach-verkettete) (Buchstaben-)Liste

a) Eine Liste hat nicht nur die vier Methoden, die es mit gleichem Namen schon bei Keller und Schlange gibt, sondern zusätzlich noch *anDenAnfang*, *weiter* und *istHintermLetzten*. Erläutere kurz, wozu bei Listen zusätzliche Methoden benötigt werden, und gib für jede der 6 Methoden außer für *zeige1* die ‚Überschrift' (Signatur) in Java sowie eine kurze Beschreibung an! (Beispiel 7: *„public char zeige1( )* nennt den aktuellen Buchstaben, ohne die Liste zu verändern")

b) Bei der Implementation der (Char-)Liste mit (Char-)Knoten hatten wir die zwei „Zeiger" *vorAnfang* und *vorAktuell* sowie ganz links einen dummy-Knoten verwendet. Erläutere, warum nicht mit *anfang* und *aktuell* (und ohne dummy-Knoten) gearbeitet werden konnte!

c) Jetzt soll die Implementation der Liste statt mit Knoten mit zwei Kellern erfolgen. Das Bild zeigt eine Liste mit den 5 Buchstaben A bis E, wobei C oben in Keller *k1* und D oben in Keller *k2* steht. Beispielhaft wird auf der nächsten Seite die Methode *weiter* genannt.



c1) Da man leicht an die oberen Buchstaben beider Keller heran kommt, könnte für *zeige1* der Liste entweder die Keller-Methode *k1.zeige1()* oder auch *k2.zeige1()* verwendet werden. Begründe im Hinblick auf die anderen Listen-Methoden, welche Variante geschickter ist!

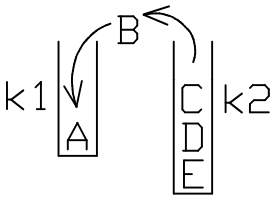
c2) Schreibe außerdem die drei Methoden *rein*, *raus* und *anDenAnfang* in dein Heft und gib ihren Aufwand im besten wie im schlimmsten Fall an (wenn die Liste enthalt insgesamt n Buchstaben enthält).

```
public class CharListe  // für 3c)
{
  CharKeller k1 = new CharKeller( );
  CharKeller k2 = new CharKeller( );

  public void weiter()
  {
    k1.rein ( k2.raus() );
  }
...
```

4 Einfache Ringliste

Beim Kartenspiel sitzen die Kinder Anna, Bernd, Claudia, Didi und Emre – kurz A, B, C, D und E – im Kreis und sind rechtsrum nacheinander dran. Ein Java-Programm soll helfen und die richtige Reihenfolge A-B-C-D-E-A-B-... ansagen. Schreibe dazu jeweils nur die Methode *public char nächstesKind()*, die immer den Kennbuchstaben des jeweils nächsten Kindes liefert (in den Bildern ist jeweils dargestellt, dass – nachdem A schon dran war – jetzt gerade Kind 'B' dran kommt)

a) Es soll bereits eine Schlange *s* geben, die mit den Buchstaben der Kinder gefüllt ist und die mit den üblichen Methoden bedient werden kann. Schreibe damit *public char nächstesKind()* (die im gezeigten Beispiel gerade den Buchstaben 'B' zurückgibt, beim nächsten Aufruf aber 'C' und bei dem nachfolgenden Aufruf 'D' liefern muss) und erkläre die Funktion!

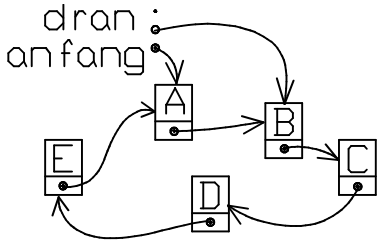


b) Jetzt sollen zur Verwaltung der Kinder die beiden Keller *k1* und *k2* verwendet werden (anfangs ist *k1* leer und *k2* bereits gefüllt. Das Bild links zeigt schon den Fall, dass 'B' dran kommt).

b1) Erläutere: Wie muss verfahren werden, wenn *k2* leer wird?

b2) Schreibe *public char nächstesKind()* unter Beachtung von b1)!

c) Nun werden die Kinder durch *CharKnoten* (vgl. Kasten zu Aufgabe 2) repräsentiert. Zwei „Zeiger" *dran* und *anfang* (beide vom Typ *CharKnoten*) seien in der Klasse *Ring* definiert.



c1) Schreibe wieder *public char nächstesKind()*, jetzt für diese Implementation (*dran* zeigt immer auf das Kind bzw. den Knoten mit dem Buchstaben, das/der gerade dran ist)

c2) Im Laufe des Spiels können Kinder ausscheiden. Schreibe die Methode *public void entferne( )*, die das Kind, das gerade dran wäre, aus dem Ring entfernt. Sein Nachfolger soll dann dran sein. Wird das Anfangs-Kind entfernt, soll auch der *anfang*-Zeiger auf den Nachfolger zeigen. Du darfst darauf vertrauen, dass *entferne* nur aufgerufen wird, wenn noch mindestens zwei Kinder spielen (und nachher also noch ein Kind übrig bleibt, das dann auf sich selbst zeigen muss und auf das *anfang* und *dran* zeigen)

d) Bei manchen Spielen – etwa Mau-Mau bzw. Uno – kann mitten im Spiel die Richtung bzw. der Drehsinn verändert werden, d.h. als nächstes wäre ab dann nicht mehr der rechte, sondern der linke Nachbar dran. Beschreibe (ohne Programmtext) für die 3 Datenstrukturen a) bis c), ob sie für einen Richtungswechsel einigermaßen geeignet wären bzw. durch welche Abänderungen der Richtungswechsel ermöglicht werden könnte.

5 Beim Mau-Mau- oder Uno-Kartenspiel liegt ein Packen Karten auf dem Tisch, von dem sich Kinder immer nur von oben eine (oder 2) Karte(n) wegnehmen können. Schreibe eine Klasse *Spielkarte* (Karten haben Farben, Werte und Punkte wie z.B. „Pik"-„As"-11) und erzeuge dann mit diesem Elementtyp einen *kartenstapel* für den Tisch (1 Zeile mit geeigneten *SpeicherMitTyp*)