Informatik Abitur Bayern 2017 / I - Lösung

Autoren:
May (1-3)
Trösch(4, 5)

1 5

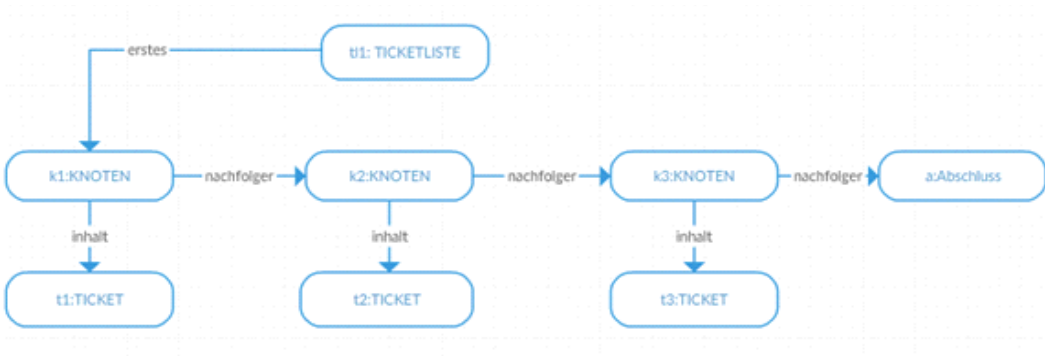
```
public class TICKET
{
  //Attribute:
  private String kennzeichen;
  private String telefonnummer;
  private int parkzeitende;

  public TICKET(String auto, String telnummer, int ende)
  {
      kennzeichen = auto;
      telefonnummer = telnummer;
      parkzeitende = ende;
  }

  public int restzeitGeben(int uhrzeit)
  {
    return parkzeitende - uhrzeit;
  }
}
```

2 Durch die SMS des Fahrers wird die Methode `registrieren(...)` eines Objekts `p` der Klasse `PARKSYSTEM` aufgerufen. Die Methode `registrieren` hat als Eingabeparameter die in der SMS enthaltenen Daten (Kennzeichen, Telefonnummer und Parkdauer). Während des Ausführens der Methode `registrieren` wird die Uhrzeit durch Aufruf von `uhrzeitGeben()` eines Objekts `u` der Klasse `UHR` bestimmt. Zudem wird ein neues Objekt `ticket1` der Klasse `TICKET` angelegt, das die geforderten Eingabeparameter Kennzeichen, Telefonnummer und Parkzeitende übergeben bekommt (Parkzeitende ist die Summe aus Parkdauer und aktueller Uhrzeit.) Das neu angelegte Ticket wird dann der Ticketliste `tl` hinzugefügt. 4

3a 5



3b In die **Klasse TICKETLISTE** müssen folgende Methoden eingefügt werden: 11

```
public void einfuegen(TICKET ticket)
{
  //Erzeugen eines Knotens mit entsprechendem Nachfolger
  //Speichern des Knotens in Variable erstes
  erstes = new KNOTEN(erstes, ticket);
}

public TICKET suchen(String kennzeichen)
{
  //Suche beginnt beim ersten Listenelement, dann
  //rekursiv weiter
  return erstes.suchen(kennzeichen);
}
```

In die (abstrakte) **Klasse LISTENELEMENT** muss folgende abstrakte Methode eingefügt werden:

```
public abstract TICKET suchen(String kennzeichen);
```

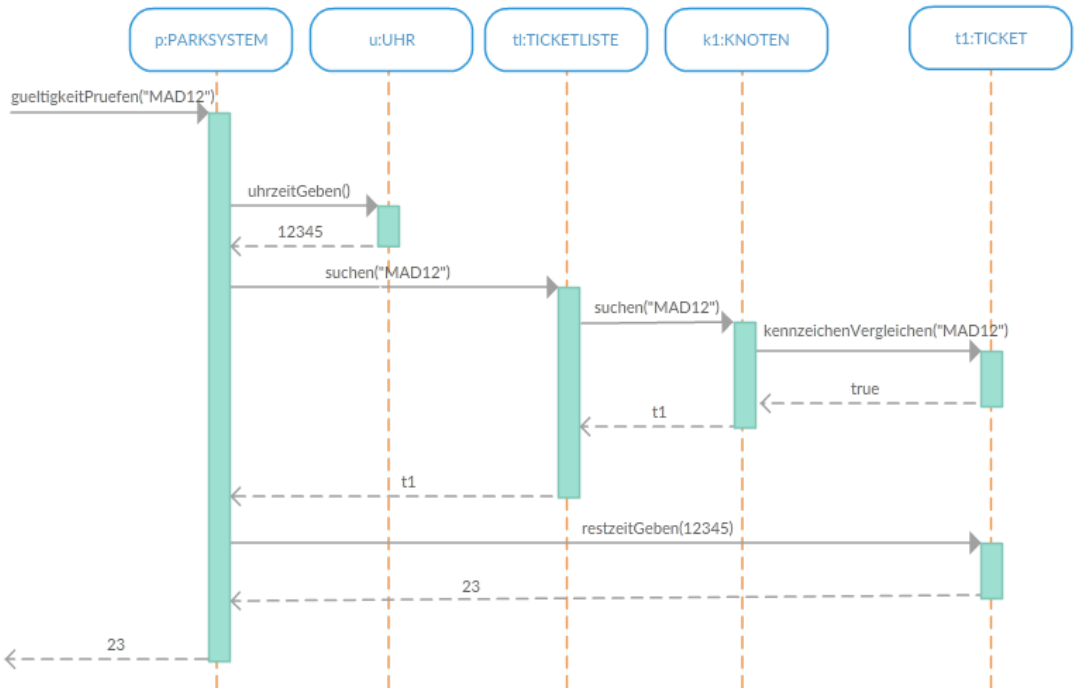
In die **Klasse KNOTEN** muss der Konstruktor und die Methode suchen() eingefügt werden:

```
public KNOTEN(LISTENELEMENT element, TICKET t)
{
  nachfolger = element;
  inhalt = t;
}
public TICKET suchen(String kennzeichen)
{
  //Stimmt das Kennzeichen überein, wird das Ticket zurückgegeben
  if(inhalt.kennzeichenVergleichen(kennzeichen))
  {
    return inhalt;
  }
  //ansonsten wird beim Nachfolger weitergesucht
  else
  {
    return nachfolger.suchen(kennzeichen);
  }
}
```

In die **Klasse ABSCHLUSS** muss ebenfalls die Methode suchen() eingefügt werden:

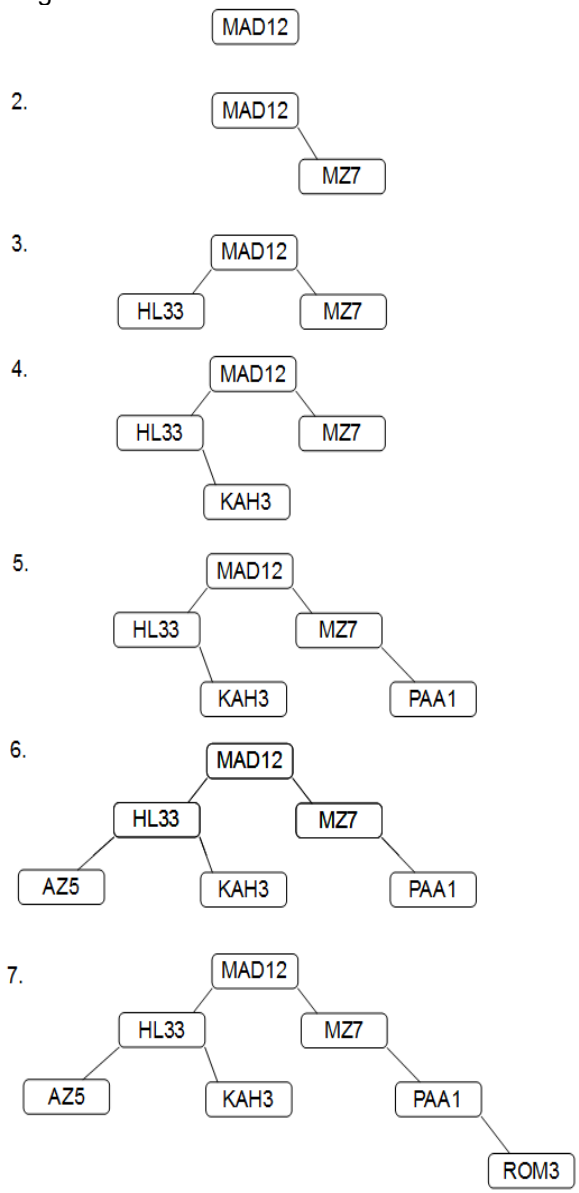
```
public TICKET suchen(String kennzeichen)
{
  //am Ende der Liste wird null zurückgegeben
  return null;
}
```

3c



3d
```
Methode int gueltigkeitPrüfen(kennzeichen)
  uz = u.uhrzeitGeben();
  t = tl.suchen(kennzeichen);
  wenn t == null dann
  {
    gib -2147483648 zurück;
  }
  sonst
  {
    gib t.restzeitGeben(uz) zurück;
  }
  endewenn
endeMethode
```
3

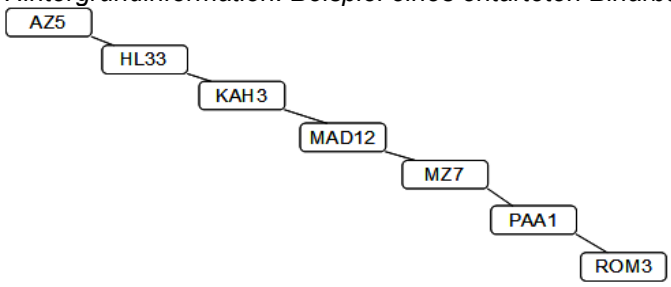
4a Der geordnete Binärbaum (alle Elemente links des jeweiligen Vaterknotens < Vaterknoten; alle Elemente rechts des jeweiligen Vaterknotens > Vaterknoten) wird folgendermaßen aufgebaut: 4



Hinweis: Für die Bearbeitung der Aufgabe reicht die Angabe des unter 7. angegebenen Binärbaumes.
Zur *Suche* eines beliebigen Kennzeichens in dem entstandenen Binärbaum sind höchstens 4 Vergleiche nötig.
(Grund: Der Baum hat die Höhe 4 und auf jeder Ebene ist in einem Binärbaum lediglich ein Vergleich notwendig.)

4b Wie in Teilaufgabe a) bereits angesprochen, ist die maximale Anzahl an Vergleichsoperationen in einem geordneten Binärbaum gleich der Baumhöhe. Der worst case ist, wenn die in den Baum einzuordnenden Elemente bereits auf- oder absteigend sortiert sind (z.B. die Reihenfolge der einzusortierenden Kennzeichen in Teilaufgabe a) lauten würde: AZ5, HL33, KAH3, MAD12, MZ7, PAA1, ROM3) und dadurch bei der Erstellung des Binärbaumes ein sogenannter entarteter Baum (entspricht einfach verketteter Liste) entsteht. 6

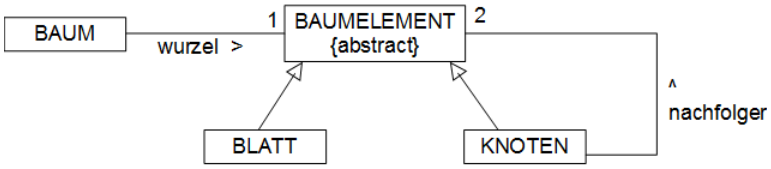
Hintergrundinformation: Beispiel eines entarteten Binärbaumes:



Die Baumhöhe entspricht hierbei der Anzahl der Elemente. Die Obergrenze der maximal benötigten Vergleiche zum Suchen in einem Baum mit 250 Elementen beträgt daher 250.

Im best case liegt ein balancierter Binärbaum vor (d.h. die Ebene der Blätter unterscheidet sich um maximal 1). In einem Binärbaum der Höhe 8 können maximal 2^8 - 1= 255 Elemente gespeichert werden.Somit sind hier im best case höchstens 8 vergleiche notwendig.

4c 2

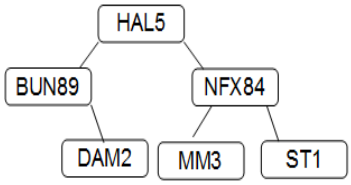


Im Vergleich zur Liste kann beim geordneten Binärbaum ein Knoten 2 Nachfolger haben.
(Hinweis: Das Zeichnen des Klassendiagramms ist zur Lösung der Aufgabe nicht unbedingt notwendig)

4d 4

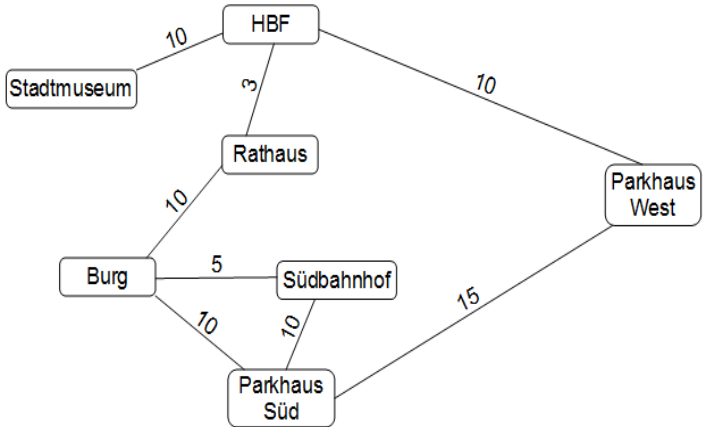
```
Methode zeigeAbgelaufeneTickets()
        linkerNF.zeigeAbgelaufeneTickets()
        wenn Ticket abgelaufen dann
              gib Ticket aus
        endewenn
        rechterNF.zeigeAbgelaufeneTickets()
endeMethode
```

4e 5



Begründung (nicht erforderlich zur Lösung der Aufgabe!)
Löschen des Tickets zum Kennzeichen GA11:
Anzahl der Kindknoten = 2
→ Suche Knoten mit kleinstem Inhalt in rechtem Teilbaum (= 'HAL5')
→ Ersetze 'GA11' durch 'HAL5'
→ Lösche ursprünglichen Knoten mit Inhalt HAL5
→ Anzahl der Kindknoten = 1
→ Ersetze Knoten 'HAL5' durch seinen Kindknoten 'MM3'

5a Graph: 9

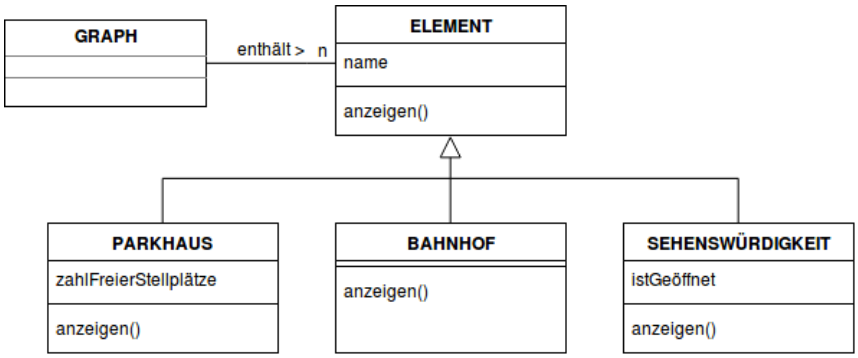


Adjazenzmatrix:

	HBF	Stadtmuseum	Rathaus	Parkhaus West	Burg	Südbahnhof	Parkhaus Süd
HBF		10	3	10	-	-	-
Stadtmuseum	10		-	-	-	-	-
Rathaus	3	-		-	10	-	-
Parkhaus West	10	-	-		-	-	15
Burg	-	-	10	-		5	10
Südbahnhof	-	-	-	-	5		10
Parkhaus Süd	-	-	-	15	10	10	

Zwei der folgenden Eigenschaften:
Die Kanten des Graphen sind gewichtet.
Der Graph enthält Zyklen.
Der Graph ist ungerichtet.
Der Graph ist zusammenhängend.

5b 5



Bemerkung: Es ist möglich, aber je nach Interpretation des beschriebenen Szenarios nicht zwingend, die Klasse ELEMENT als abstrakt zu deklarieren.

5c 9

```
Methode zeigeAlleStandorteMitMax1Umstieg(KNOTEN k)
        wiederhole für alle Nachbarknoten k1 von k
                fahrzeit1 = Kantengewicht Kante k_k1
                gib Bezeichnung von k1 und fahrzeit1 aus
                wiederhole für alle Nachbarknoten k2 von k1
                        wenn k2 ist nicht k dann
                        fahrzeit2 = fahrzeit1 + 5 + Kantengewicht Kante k1_k2
                        gib Bezeichnung von k2 und fahrzeit2 aus
                        endewenn
                endewiederhole
        endewiederhole
endeMethode
```