Name: _____

Abiturprüfung 2021

Informatik, Leistungskurs

Aufgabenstellung:

Ein junges Startup-Unternehmen verleiht in Köln Elektro-Roller, sogenannte Scooter. Alle Scooter sind mit einem GPS-Empfänger ausgestattet und übermitteln über eine Internetverbindung im Minutentakt ihre aktuelle Position (Standort) sowie den aktuellen Batterie-Zustand (Akkuladung) an den Server des Unternehmens. Um den Abstand eines Scooters zur Firmenzentrale leichter berechnen zu können, übermitteln die Scooter ihre Position relativ zur Firmenzentrale des Unternehmens (Koordinatenursprung). Die jeweils letzte Position wird serverseitig als Punkt in einem Koordinatensystem gespeichert. Abbildung 1 zeigt die aktuelle Situation im näheren Umfeld der Firmenzentrale.

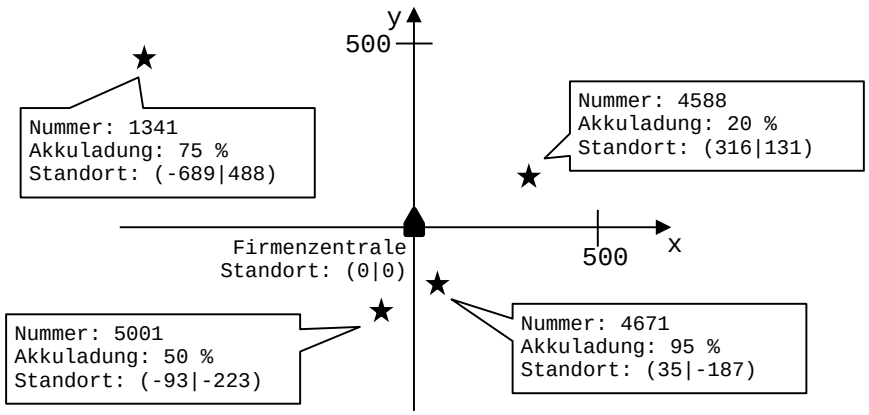


Abbildung 1: Aktuelle Positionen der Scooter in der Nähe der Firmenzentrale

Das Unternehmen hat sich dazu entschieden, die Scooter in einem binären Suchbaum zu verwalten. Die eindeutige vierstellige (Serien-)Nummer der Scooter dient dazu als Suchschlüssel innerhalb des Baums.

Das Unternehmen nutzt bereits die vier in Abbildung 1 dargestellten Scooter. Der sich daraus ergebende binäre Suchbaum ist in Abbildung 2 dargestellt.

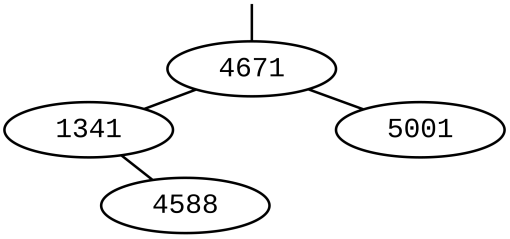


Abbildung 2: Suchbaum mit den Suchschlüsseln der vier Scooter

Name: _____

a) Das Unternehmen besitzt noch mehr Scooter, welche in den vorhandenen Datenbestand eingefügt werden sollen. Zunächst werden vier Scooter mit den Nummern `4999`, `4400`, `1000` und `8500` in genau dieser Reihenfolge in den Suchbaum aus Abbildung 2 eingefügt.

Stellen Sie den nach dem Einfügen entstandenen Suchbaum grafisch dar.

Stellen Sie den Suchbaum grafisch dar, der aus dem Suchbaum mit den acht Suchschlüsseln entsteht, wenn der Scooter mit der Nummer `4671` *gestohlen wurde und deshalb aus dem System gelöscht werden muss.*

(7 Punkte)

Das Unternehmen verwaltet mithilfe seiner Software neben den Daten der Scooter auch die Daten der Kundinnen und Kunden, die einen Scooter ausleihen möchten. Aus Gründen der Vereinfachung ist dabei die Verwaltung so modelliert, dass jede Kundin und jeder Kunde lediglich durch eine eindeutige E-Mail-Adresse und eine Kreditkartennummer zu Abrechnungszwecken beschrieben wird.

Die Software wurde auf Basis des in Abbildung 3 dargestellten Ausschnitts des Implementationsdiagramms entwickelt.

Name: _____

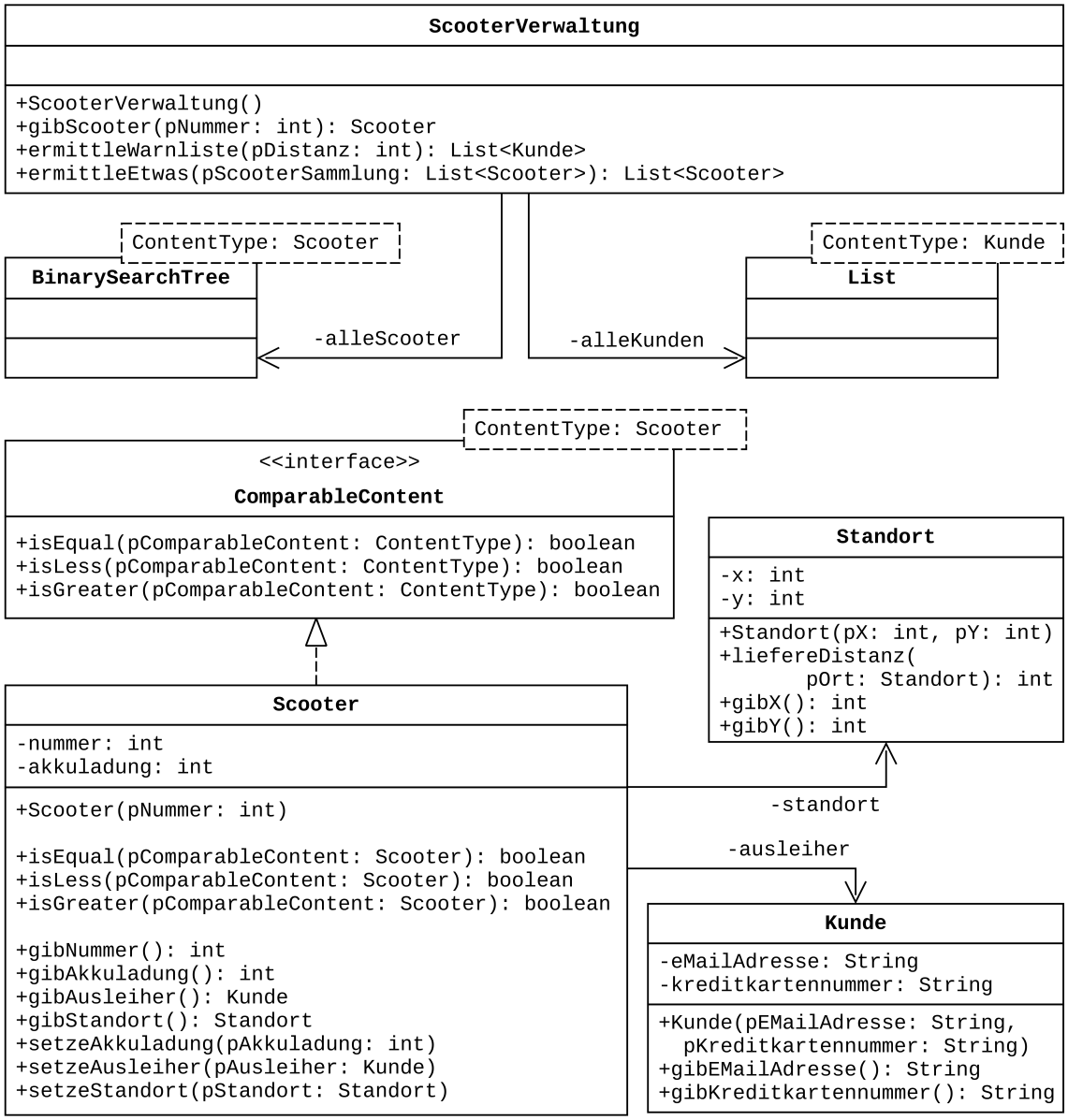


Abbildung 3: Ausschnitt des Implementationsdiagramms

b) *Erläutern Sie die Beziehungen zwischen den Klassen* `ScooterVerwaltung`, `Scooter`, `Kunde`, `List` *und* `BinarySearchTree` *sowie dem Interface* `ComparableContent`.

Begründen Sie im Sachkontext, weshalb die Verwaltung der Scooter in Form eines binären Suchbaums effizienter ist als die Verwaltung in einer linearen Liste, wohingegen die Verwaltung der Kundinnen und Kunden in Form einer Liste vertretbar ist.

(8 Punkte)

Name: _____

Jeden Abend muss überprüft werden, ob alle Scooter vorschriftsmäßig geparkt wurden. Zu diesem Zweck hat sich das Unternehmen eine Drohne angeschafft, welche abends alle Scooter abfliegt und jeweils ein Foto des Scooters an die Firmenzentrale schickt.

c) Ein Praktikant hat die folgende Methode der Klasse `ScooterVerwaltung` implementiert.

```
 1 public List<Scooter> ermittleEtwas(
 2                             List<Scooter> pScooterSammlung) {
 3   Standort ort = new Standort(0, 0);
 4   List<Scooter> ergebnis = new List<Scooter>();
 5   while (!pScooterSammlung.isEmpty()) {
 6     pScooterSammlung.toFirst();
 7     Scooter naechster = pScooterSammlung.getContent();
 8     pScooterSammlung.next();
 9     while (pScooterSammlung.hasAccess()) {
10       Scooter test = pScooterSammlung.getContent();
11       if (ort.liefereDistanz(test.gibStandort())
12               < ort.liefereDistanz(naechster.gibStandort())) {
13         naechster = test;
14       }
15       pScooterSammlung.next();
16     }
17     pScooterSammlung.toFirst();
18     while (pScooterSammlung.getContent() != naechster) {
19       pScooterSammlung.next();
20     }
21     pScooterSammlung.remove();
22     ergebnis.append(naechster);
23     ort = naechster.gibStandort();
24   }
25   return ergebnis;
26 }
```

Die Methode `ermittleEtwas` werde mit dem Parameter `pScooterSammlung` aufgerufen, welcher eine Liste referenziert, die die in Abbildung 1 dargestellten Scooter in der Reihenfolge `4588`, `4671`, `1341` und `5001` enthält.

Analysieren Sie den Quelltext der Methode, indem Sie sie auf die Beispieldaten anwenden und die so erzeugte Rückgabe-Liste ermitteln.

Erläutern Sie die Funktionsweise der Methode.

Erläutern Sie im Sachkontext, welche Information diese Methode liefert.

(12 Punkte)

Name: _____

d) Das allabendliche Anfliegen der Scooter durch eine Drohne wird problematisch, wenn sich die Scooter zu weit von der Firmenzentrale entfernt befinden. Deshalb sollen Kundinnen und Kunden, die sich z. B. mehr als 10 000 Meter von der Firmenzentrale entfernen, über ihre hinterlegte E-Mail-Adresse eine Warnmeldung erhalten.

Zur Ermittlung der Liste aller Personen, die eine solche Warnmeldung erhalten sollen, stellt die Klasse `ScooterVerwaltung` die Methode `ermittleWarnliste` zur Verfügung, welche den folgenden Methodenkopf besitzt:

```
public List<Kunde> ermittleWarnliste(int pDistanz)
```

Die Dokumentation dieser Methode ist in der Anlage zu finden.

Entwickeln Sie eine Strategie für einen Algorithmus dieser Methode.

Implementieren Sie die Methode entsprechend der Dokumentation.

(11 Punkte)

e) Ein Praktikant bemerkt, dass die Entfernung eines Scooters von der Firmenzentrale eine sehr wichtige Information ist, welche in verschiedenen Kontexten benötigt wird, und nennt hierfür zwei Beispiele:

- Für entliehene Scooter ist diese Entfernung zum Beispiel bei der Versendung von Warnmeldungen für zu weit entfernte Scooter wichtig.
- Für nicht entliehene Scooter ist diese Entfernung dagegen ein wichtiges Kriterium für den Flug der Drohne, die das vorschriftsmäßige Parken der Scooter überprüft.

Er schlägt deshalb vor, diese Entfernung zwischen Scooter und Firmenzentrale als Sortierkriterium in dem binären Suchbaum zu verwenden.

Erläutern Sie, welche Veränderungen in der Modellierung vorgenommen werden müssen, um den Vorschlag des Praktikanten umzusetzen.

Begründen Sie, warum der Vorschlag des Praktikanten problematisch ist, wenn verschiedene Scooter die gleiche Entfernung zur Firmenzentrale haben.

Beurteilen Sie mit Bezug auf die beiden vom Praktikanten genannten Anwendungsbeispiele, inwiefern der Modellierungsvorschlag des Praktikanten sinnvoll ist.

(12 Punkte)

Zugelassene Hilfsmittel:

- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse `Scooter`

Objekte dieser Klasse verwalten die Daten eines Scooters.

Dokumentation der Klasse `Scooter`

`Scooter(int pNummer)`

Ein Objekt der Klasse wird initialisiert. Die eindeutige Nummer `pNummer` wird gespeichert.

`boolean isEqual(Scooter pComparableContent)`

Wenn die eindeutige Nummer des `Scooter`-Objekts, von dem die Methode aufgerufen wird, gleich der Nummer des `Scooter`-Objekts `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

`boolean isLess(Scooter pComparableContent)`

Wenn die eindeutige Nummer des `Scooter`-Objekts, von dem die Methode aufgerufen wird, kleiner als die Nummer des `Scooter`-Objekts `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

`boolean isGreater(Scooter pComparableContent)`

Wenn die eindeutige Nummer des `Scooter`-Objekts, von dem die Methode aufgerufen wird, größer als die Nummer des `Scooter`-Objekts `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

`int gibNummer()`

Die Methode liefert die eindeutige Nummer des Scooters.

`int gibAkkuladung()`

Die Methode liefert den Batterie-Zustand des Scooters. Der Wert entspricht der Akkuladung in Prozent, d. h., der Wert liegt im Wertebereich `0` (Akku ist leer) bis `100` (Akku ist vollständig aufgeladen).

`Kunde gibAusleiher()`

Die Methode liefert das Objekt der Klasse `Kunde`, welcher den Scooter momentan ausgeliehen hat. Ist der Scooter momentan nicht verliehen, so wird `null` zurückgegeben.

`Standort gibStandort()`

Die Methode liefert ein Objekt der Klasse `Standort`, welches die Koordinaten relativ zur Firmenzentrale enthält. Z. B. bedeutet der Standort `(-300|400)`, dass sich der Scooter 300 Meter in westlicher Richtung und 400 Meter in nördlicher Richtung von der Firmenzentrale befindet.

Name: _____

`void setzeAkkuladung(int pAkkuladung)`

Die Methode verändert den Wert der Akkuladung auf den Wert `pAkkuladung`. Ist der Wert kleiner als `0` oder größer als `100`, so geschieht nichts.

`void setzeAusleiher(Kunde pAusleiher)`

Die Methode speichert die Daten der Person, die den Scooter momentan ausgeliehen hat. Ist der Parameter `pAusleiher` gleich `null`, so ist der Scooter wieder frei und kann von jemand anderem ausgeliehen werden.

`void setzeStandort(Standort pStandort)`

Die Methode setzt den Standort des Scooters auf `pStandort`.

Die Klasse `ScooterVerwaltung`

Objekte dieser Klasse verwalten Scooter sowie Kundinnen und Kunden und ermöglichen die Ausleihe von Scootern.

Dokumentation der Klasse `ScooterVerwaltung`

`ScooterVerwaltung()`

Ein Objekt der Klasse wird initialisiert. Es verwaltet noch keine Scooter und keine Daten von Kundinnen und Kunden.

`Scooter gibScooter(int pScooternummer)`

Liefert das `Scooter`-Objekt mit der (Serien-)Nummer `pScooternummer`. Falls es kein `Scooter`-Objekt mit der entsprechenden (Serien-)Nummer gibt, dann wird `null` zurückgeliefert.

`List<Kunde> ermittleWarnliste(int pDistanz)`

Die Methode liefert eine Liste aller Kundinnen und Kunden, die sich mit ihrem ausgeliehenen Scooter um mehr als `pDistanz` Meter Luftlinie von der Firmenzentrale entfernt haben. Die Firmenzentrale befindet sich im Standort `(0|0)`.
Gibt es keine Kundin und keinen Kunden, die bzw. der sich zu weit entfernt hat, oder ist der Parameter `pDistanz < 0`, so wird eine leere Liste zurückgegeben.
Scooter, die momentan nicht ausgeliehen wurden, sich aber in größerer Distanz befinden, werden nicht berücksichtigt, weil es keine Person gibt, an den eine Warnmeldung verschickt werden könnte.

`List<Scooter> ermittleEtwas(List<Scooter> pScooterSammlung)`

Die Methode wird in Teilaufgabe c) analysiert.

Name: _____

Die Klasse `Kunde`

Objekte dieser Klasse verwalten die Daten einer Kundin bzw. eines Kunden.

Dokumentation der Klasse `Kunde`

`Kunde(String pEMailAdresse, String pKreditkartennummer)`

Ein Objekt der Klasse wird initialisiert. Die eindeutige E-Mail-Adresse `pEMailAdresse` der Kundin bzw. des Kunden und die zugehörige Kreditkartennummer `pKreditkartennummer` werden gespeichert.

`String gibEMailAdresse()`

Die eindeutige E-Mail-Adresse der Kundin bzw. des Kunden wird zurückgeliefert.

`String gibKreditkartennummer()`

Die Kreditkartennummer der Kundin bzw. des Kunden wird zurückgeliefert.

Die Klasse `Standort`

Objekte dieser Klasse verwalten die Koordinaten eines Standorts für Scooter. Die Koordinaten werden relativ zur Firmenzentrale gespeichert, welche sich im Koordinatenursprung befindet.

Dokumentation der Klasse `Standort`

`Standort(int pX, int pY)`

Ein Objekt der Klasse wird initialisiert. Die Koordinaten (`pX`|`pY`) werden im Objekt gespeichert.

`int liefereDistanz(Standort pOrt)`

Die auf volle Meter abgerundete Distanz (Luftlinie in Metern) zwischen dem gespeicherten Ort und dem durch `pOrt` referenzierten Ort wird zurückgeliefert. Sollte `pOrt` den Wert `null` haben, so wird der Wert `0` zurückgeliefert.

`int gibX()`

Die x-Koordinate des Standorts wird zurückgeliefert. Diese entspricht der Entfernung in westlicher (bei negativem Wert) bzw. in östlicher (bei positivem Wert) Richtung in Metern von der Firmenzentrale.

`int gibY()`

Die y-Koordinate des Standorts wird zurückgeliefert. Diese entspricht der Entfernung in südlicher (bei negativem Wert) bzw. in nördlicher (bei positivem Wert) Richtung in Metern von der Firmenzentrale.

Name: _____

Die generische Klasse `List`

Objekte der generischen Klasse `List` verwalten beliebig viele, linear angeordnete Objekte vom Typ `ContentType`. Auf höchstens ein Listenobjekt, aktuelles Objekt genannt, kann jeweils zugegriffen werden. Wenn eine Liste leer ist, vollständig durchlaufen wurde oder das aktuelle Objekt am Ende der Liste gelöscht wurde, gibt es kein aktuelles Objekt. Das erste oder das letzte Objekt einer Liste kann durch einen Auftrag zum aktuellen Objekt gemacht werden. Außerdem kann das dem aktuellen Objekt folgende Listenobjekt zum neuen aktuellen Objekt werden.
Das aktuelle Objekt kann gelesen, verändert oder gelöscht werden. Außerdem kann vor dem aktuellen Objekt ein Listenobjekt eingefügt werden.

Dokumentation der Klasse `List<ContentType>`

`List()`

Eine leere Liste wird erzeugt.

`boolean isEmpty()`

Die Anfrage liefert den Wert `true`, wenn die Liste keine Objekte enthält, sonst liefert sie den Wert `false`.

`boolean hasAccess()`

Die Anfrage liefert den Wert `true`, wenn es ein aktuelles Objekt gibt, sonst liefert sie den Wert `false`.

`void next()`

Falls die Liste nicht leer ist, es ein aktuelles Objekt gibt und dieses nicht das letzte Objekt der Liste ist, wird das dem aktuellen Objekt in der Liste folgende Objekt zum aktuellen Objekt, andernfalls gibt es nach Ausführung des Auftrags kein aktuelles Objekt, d. h., `hasAccess()` liefert den Wert `false`.

`void toFirst()`

Falls die Liste nicht leer ist, wird das erste Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`void toLast()`

Falls die Liste nicht leer ist, wird das letzte Objekt der Liste aktuelles Objekt. Ist die Liste leer, geschieht nichts.

`ContentType getContent()`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt zurückgegeben. Andernfalls (`hasAccess() == false`) gibt die Anfrage den Wert `null` zurück.

Name: _____

`void setContent(ContentType pContent)`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`) und `pContent` ungleich `null` ist, wird das aktuelle Objekt durch `pContent` ersetzt. Sonst bleibt die Liste unverändert.

`void append(ContentType pContent)`

Ein neues Objekt `pContent` wird am Ende der Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Wenn die Liste leer ist, wird das Objekt `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt (`hasAccess() == false`). Falls `pContent` gleich `null` ist, bleibt die Liste unverändert.

`void insert(ContentType pContent)`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird ein neues Objekt `pContent` vor dem aktuellen Objekt in die Liste eingefügt. Das aktuelle Objekt bleibt unverändert. Falls die Liste leer ist und es somit kein aktuelles Objekt gibt (`hasAccess() == false`), wird `pContent` in die Liste eingefügt und es gibt weiterhin kein aktuelles Objekt. Falls es kein aktuelles Objekt gibt (`hasAccess() == false`) und die Liste nicht leer ist oder `pContent == null` ist, bleibt die Liste unverändert.

`void concat(List<ContentType> pList)`

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls es sich bei der Liste und `pList` um dasselbe Objekt handelt, `pList == null` oder eine leere Liste ist, bleibt die Liste unverändert.

`void remove()`

Falls es ein aktuelles Objekt gibt (`hasAccess() == true`), wird das aktuelle Objekt gelöscht und das Objekt hinter dem gelöschten Objekt wird zum aktuellen Objekt. Wird das Objekt, das am Ende der Liste steht, gelöscht, gibt es kein aktuelles Objekt mehr (`hasAccess() == false`). Wenn die Liste leer ist oder es kein aktuelles Objekt gibt (`hasAccess() == false`), bleibt die Liste unverändert.

Name: _____

Die generische Klasse `BinarySearchTree`

Mithilfe der generischen Klasse `BinarySearchTree` können beliebig viele Objekte des Typs `ContentType` in einem Binärbaum (binärer Suchbaum) entsprechend einer Ordnungsrelation verwaltet werden.
Ein Objekt der Klasse `BinarySearchTree` stellt entweder einen leeren Baum dar oder verwaltet ein Inhaltsobjekt vom Typ `ContentType` sowie einen linken und einen rechten Teilbaum, die ebenfalls Objekte der Klasse `BinarySearchTree` sind.
Die Klasse der Objekte, die in dem Suchbaum verwaltet werden sollen, muss das generische Interface `ComparableContent` implementieren. Dabei muss durch Überschreiben der drei Vergleichsmethoden `isLess`, `isEqual`, `isGreater` (siehe Dokumentation des Interfaces) eine eindeutige Ordnungsrelation festgelegt sein.

Die Objekte der Klasse `ContentType` sind damit vollständig geordnet. Für je zwei Objekte `c1` und `c2` vom Typ `ContentType` gilt also insbesondere genau eine der drei Aussagen:

- `c1.isLess(c2)` (Sprechweise: `c1` ist kleiner als `c2`)
- `c1.isEqual(c2)` (Sprechweise: `c1` ist gleichgroß wie `c2`)
- `c1.isGreater(c2)` (Sprechweise: `c1` ist größer als `c2`)

Alle Objekte im linken Teilbaum sind kleiner als das Inhaltsobjekt des Binärbaumes. Alle Objekte im rechten Teilbaum sind größer als das Inhaltsobjekt des Binärbaumes. Diese Bedingung gilt auch in beiden Teilbäumen.

Name: _____

Dokumentation der generischen Klasse `BinarySearchTree<ContentType extends ComparableContent<ContentType>>`

`BinarySearchTree()`

Der Konstruktor erzeugt einen leeren Suchbaum.

`boolean isEmpty()`

Diese Anfrage liefert den Wahrheitswert `true`, wenn der Suchbaum leer ist, sonst liefert sie den Wert `false`.

`void insert(ContentType pContent)`

Falls bereits ein Objekt in dem Suchbaum vorhanden ist, das gleichgroß ist wie `pContent`, passiert nichts. Andernfalls wird das Objekt `pContent` entsprechend der Ordnungsrelation in den Baum eingeordnet. Falls der Parameter `null` ist, ändert sich nichts.

`ContentType search(ContentType pContent)`

Falls ein Objekt im binären Suchbaum enthalten ist, das gleichgroß ist wie `pContent`, liefert die Anfrage dieses, ansonsten wird `null` zurückgegeben. Falls der Parameter `null` ist, wird `null` zurückgegeben.

`void remove(ContentType pContent)`

Falls ein Objekt im binären Suchbaum enthalten ist, das gleichgroß ist wie `pContent`, wird dieses entfernt. Falls der Parameter `null` ist, ändert sich nichts.

`ContentType getContent()`

Diese Anfrage liefert das Inhaltsobjekt des Suchbaumes. Wenn der Suchbaum leer ist, wird `null` zurückgegeben.

`BinarySearchTree<ContentType> getLeftTree()`

Diese Anfrage liefert den linken Teilbaum des binären Suchbaumes. Der binäre Suchbaum ändert sich nicht. Wenn er leer ist, wird `null` zurückgegeben.

`BinarySearchTree<ContentType> getRightTree()`

Diese Anfrage liefert den rechten Teilbaum des Suchbaumes. Der Suchbaum ändert sich nicht. Wenn er leer ist, wird `null` zurückgegeben.

Name: _____

Das generische Interface `ComparableContent<ContentType>`

Das generische Interface `ComparableContent` muss von Klassen implementiert werden, deren Objekte in einen Suchbaum (BinarySearchTree) eingefügt werden sollen. Die Ordnungsrelation wird in diesen Klassen durch Überschreiben der drei implizit abstrakten Methoden `isGreater`, `isEqual` und `isLess` festgelegt.

Das Interface `ComparableContent` gibt folgende implizit abstrakte Methoden vor:

`boolean isGreater(ContentType pComparableContent)`

Wenn festgestellt wird, dass das Objekt, von dem die Methode aufgerufen wird, bzgl. der gewünschten Ordnungsrelation größer als das Objekt `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

`boolean isEqual(ContentType pComparableContent)`

Wenn festgestellt wird, dass das Objekt, von dem die Methode aufgerufen wird, bzgl. der gewünschten Ordnungsrelation gleich dem Objekt `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

`boolean isLess(ContentType pComparableContent)`

Wenn festgestellt wird, dass das Objekt, von dem die Methode aufgerufen wird, bzgl. der gewünschten Ordnungsrelation kleiner als das Objekt `pComparableContent` ist, wird `true` geliefert. Sonst wird `false` geliefert.

Unterlagen für die Lehrkraft

Abiturprüfung 2021

Informatik, Leistungskurs

1. Aufgabenart

Analyse, Modellierung und Implementation von kontextbezogenen Problemstellungen mit Schwerpunkt auf den Inhaltsfeldern Daten und ihre Strukturierung und Algorithmen

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2021 (Stand: August 2020)

Die Aufgaben weisen vielfältige Bezüge zu den Kompetenzerwartungen und Inhaltsfeldern des Kernlehrplans bzw. zu den in den Vorgaben ausgewiesenen Fokussierungen auf.
Im Folgenden wird auf Bezüge von zentraler Bedeutung hingewiesen.

1. *Inhaltsfelder und inhaltliche Schwerpunkte*
   Daten und ihre Strukturierung
   - Objekte und Klassen
     - Entwurfsdiagramme und Implementationsdiagramme
     - Lineare Strukturen
       Lineare Liste (Klasse `List`*)*
     - Nicht-lineare Strukturen
       Binärer Suchbaum (Klasse `BinarySearchTree`*)*

   Algorithmen
   - Analyse, Entwurf und Implementierung von Algorithmen
   - Algorithmen in ausgewählten informatischen Kontexten

   Formale Sprachen und Automaten
   - Syntax und Semantik einer Programmiersprache
     - Java

2. *Medien/Materialien*
   - entfällt

[1] Die Aufgabenstellung deckt inhaltlich alle drei Anforderungsbereiche ab.

5. Zugelassene Hilfsmittel

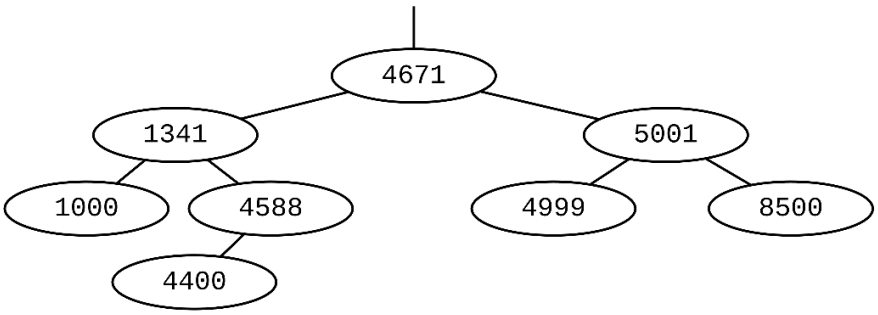
- GTR (grafikfähiger Taschenrechner) oder CAS (Computer-Algebra-System)
- Wörterbuch zur deutschen Rechtschreibung

6. Modelllösungen

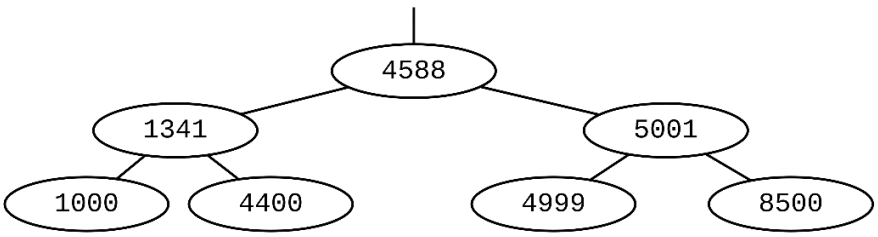
Die jeweilige Modelllösung stellt eine mögliche Lösung bzw. Lösungsskizze dar. Der gewählte Lösungsansatz und -weg der Schülerinnen und Schüler muss nicht identisch mit dem der Modelllösung sein. Sachlich richtige Alternativen werden mit entsprechender Punktzahl bewertet (Bewertungsbogen: Zeile „Sachlich richtige Lösungsalternative zur Modelllösung“).

Teilaufgabe a)

Der folgende Suchbaum entspricht den Anforderungen nach dem Einfügen der Suchschlüssel `4999`, `4400`, `1000` und `8500` in den Suchbaum.



Der folgende Suchbaum entspricht den Anforderungen nach dem Löschen des Suchschlüssels `4671` aus dem Suchbaum.



Teilaufgabe b)

Ein Objekt der Klasse `ScooterVerwaltung` verwaltet beliebig viele Objekte der Klasse `Kunde` in einem durch `Kunde` parametrisierten Objekt der Klasse `List`, welche durch das Attribut `alleKunden` referenziert wird. Zudem verwaltet ein Objekt der Klasse `ScooterVerwaltung` beliebig viele Objekte der Klasse `Scooter` in einem durch `Scooter` parametrisierten Objekt der Klasse `BinarySearchTree`, welches durch das Attribut `alleScooter` referenziert wird.

Die Klasse `Scooter` implementiert das mit `Scooter` parametrisierte Interface `ComparableContent`. Somit können Objekte der Klasse `Scooter` in einen binären Suchbaum eingefügt werden. Außerdem verwaltet ein Objekt der Klasse `Scooter` die Daten der Person, die den Scooter entliehen hat, in einem Objekt der Klasse `Kunde`.

Im Sachkontext ist eine Verwaltung der Scooter in einem binären Suchbaum deshalb sinnvoll, da die Scooter im Minutentakt ihren aktuellen Standort senden, welcher dann in den Objekten der Klasse `Scooter` aktualisiert werden muss. Es ist also eine sehr häufige Suche nach einem `Scooter`-Objekt nötig, weshalb der binäre Suchbaum aus Laufzeitgründen zu bevorzugen ist.

Die Verwaltung der Kundinnen und Kunden kann in Form einer Liste erfolgen, da lediglich beim Prozess des Ausleihens eine Kundin bzw. ein Kunde gesucht werden muss. Die Häufigkeit der Suche nach einer Kundin bzw. einem Kunden ist also wesentlich geringer als die Häufigkeit der Suche nach einem Scooter.

Teilaufgabe c)

Die Methode wird mit der Liste `pScootersammlung` aufgerufen, welche die `Scooter`-Objekte in der Reihenfolge `[4588, 4671, 1341, 5001]` enthält. Die Methode ermittelt dann wie folgt die Rückgabe-Liste:

Schritt/Zeile	`ort`	`pScooterSammlung`	`ergebnis`	`naechster`
Initialisierung Zeile 1 bis 4	`(0|0)`	`[4588, 4671, 1341, 5001]`	`[]`	
Zeile 7				`4588`
Schleife Zeile 9 bis 16				`4671`
Zeile 17 bis 23	`(35|-187)`	`[4588, 1341, 5001]`	`[4671]`	
Zeile 7				`4588`
Schleife Zeile 9 bis 16				`5001`
Zeile 17 bis 23	`(-93|-223)`	`[4588, 1341]`	`[4671, 5001]`	
Zeile 7				`4588`
Schleife Zeile 9 bis 16				`4588`
Zeile 17 bis 23	`(316|131)`	`[1341]`	`[4671, 5001, 4588]`	
Zeile 7				`1341`
Zeile 21 bis 23	`(-689|488)`	`[]`	`[4671, 5001, 4588, 1341]`	

Zum Schluss wird die Liste `ergebnis` mit den `Scooter`-Objekten in der Reihenfolge `[4671, 5001, 4588, 1341]` zurückgegeben.

Die Funktionsweise der Methode entspricht einer Umsortierung der Liste `pScooterSammlung` in einer neuen Liste, welche zurückgeliefert wird. Nacheinander werden die `Scooter`-Objekte der Liste `pScooterSammlung` entfernt und in einer neuen Reihenfolge in die Ergebnisliste eingefügt.

Die Methode ermittelt eine Liste von Scootern, welche die gleichen Scooter enthält, wie die Liste des Parameters `pScooterSammlung`, allerdings in einer bezüglich der Problemstellung angepassten Reihenfolge. Die Reihenfolge ist dabei so gewählt, dass der erste Scooter derjenige ist, der der Firmenzentrale am nächsten liegt. Gibt es mehrere Scooter, die gleich weit entfernt liegen, so wird der erste Scooter der Liste `pScooterSammlung` gewählt. Alle weiteren Scooter sind so sortiert, dass sie jeweils immer der nächstgelegene Scooter zum vorherigen sind. Auch hier gilt, dass bei gleich weit entfernten Scootern immer derjenige zuerst gewählt wird, der in der Liste `pScooterSammlung` weiter vorne stand. Im Sachkontext wird damit also eine Besuchsreihenfolge ermittelt, welche die abzufliegende Gesamtstrecke für das allabendliche Anfliegen der Scooter ausgehend von der Firmenzentrale zu reduzieren versucht.

Teilaufgabe d)

Eine Strategie des Algorithmus lautet wie folgt:

- Durchlaufe den binären Suchbaum rekursiv und besuche jeden Knoten des Baums.
- Überprüfe innerhalb eines jeden Rekursionsschritts, ob der Baum leer ist. Ist dies der Fall, so ist die Rückgabe-Liste ebenfalls leer. Andernfalls wird das `Scooter`-Objekt der Wurzel des Baumes überprüft.
- Prüfe, ob dessen Distanz zur Firmenzentrale größer als der Parameter `pDistanz` ist und der Scooter ausgeliehen ist. Ist dies der Fall, so erstelle eine Liste mit diesem `Scooter`-Objekt als einzigem Objekt. Andernfalls erstelle eine leere Liste.
- Erstelle durch Rekursionsaufruf jeweils eine Liste für den linken Teilbaum und eine für den rechten Teilbaum.
- Erstelle die gesamte Rückgabe-Liste durch Aneinanderreihung der drei zuvor erstellten Listen und gib diese zurück.

Eine mögliche Implementierung lautet wie folgt:

```
public List<Kunde> ermittleWarnliste(int pDistanz) {
  return ermittleWarnliste(pDistanz, alleScooter);
}

private List<Kunde> ermittleWarnliste(
                    int pDistanz, BinarySearchTree<Scooter> pBaum) {
  List<Kunde> ergebnis = new List<Kunde>();
  if (!pBaum.isEmpty()) {
    Scooter s = pBaum.getContent();
    if (s.gibStandort().liefereDistanz(new Standort(0,0))
                          > pDistanz && s.gibAusleiher() != null) {
      ergebnis.append(s.gibAusleiher());
    }
    ergebnis.concat(ermittleWarnliste(
                                    pDistanz, pBaum.getLeftTree()));
    ergebnis.concat(ermittleWarnliste(
                                   pDistanz, pBaum.getRightTree()));
  }
  return ergebnis;
}
```

Teilaufgabe e)

Die bisherige Modellierung müsste wie folgt verändert werden:
Die Entfernung eines Scooters von der Firmenzentrale sollte eine Eigenschaft eines Objekts der Klasse `Scooter` sein, um so als Suchschlüssel verwendet zu werden. Die Klasse `Scooter` könnte also zum Beispiel um ein Attribut `entfernung` erweitert werden. Weitere Veränderungen der Modellierung sind prinzipiell nicht nötig, lediglich die Funktionalität der von `ComparableContent` geerbten Vergleichsmethoden muss dahingehend angepasst werden, dass nun die Entfernung zur Firmenzentrale beider `Scooter`-Objekte verglichen wird.

Die Speicherung verschiedener Scooter mit gleicher Entfernung ist in dieser Modellierung problematisch, da der Suchbaum nur Objekte mit verschiedenen Suchschlüsseln aufnehmen kann. Wenn also in den von der Klasse `ComparableContent` geerbten Vergleichsmethoden nur die Werte des Attributs `entfernung` beider `Scooter`-Objekte verglichen werden, so könnten keine zwei Scooter mit gleicher Entfernung im Baum gespeichert werden.

Für entliehene Scooter ist der Modellierungsvorschlag ungeeignet, da die Scooter in Bewegung sind und ihre aktuelle Position im Minutentakt an die Software senden. Deshalb müsste das Attribut `entfernung` aller entliehenen `Scooter`-Objekte im Minutentakt verändert werden. Damit einhergehend müssten dann aber auch alle diese `Scooter`-Objekte in dem binären Suchbaum umsortiert werden.

Für nicht entliehene Scooter entfällt das vorherige Argument, allerdings ist der Modellierungsvorschlag für den dargestellten Anwendungsfall ebenfalls nicht geeignet. Dies liegt daran, dass in der Regel nicht nach einem konkreten Suchschlüssel (also einer konkreten Entfernung) gesucht wird, sondern z. B. nach „der kleinsten Entfernung“ oder nach „einer Entfernung größer als 10 000 Meter“. Für solche Suchanfragen bietet der binäre Suchbaum allerdings keine Vorteile.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings:_____ Kursbezeichnung:_____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK[2]**	**ZK**	**DK**
1	stellt den Suchbaum grafisch dar, der sich nach dem Einfügen ergibt.	4			
2	stellt den Suchbaum grafisch dar, der sich nach dem Löschen ergibt.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …… ……					
	Summe Teilaufgabe a)	**7**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert die Beziehungen zwischen den angegebenen Klassen und dem Interface.	5			
2	begründet im Sachkontext, weshalb die Verwaltung der Scooter in Form eines binären Suchbaums effizienter ist als die Verwaltung in einer linearen Liste, wohingegen die Verwaltung der Kundinnen und Kunden in einer linearen Liste vertretbar ist.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe b)	**8**			

[2] EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert den Quelltext der Methode, indem er sie auf die Beispieldaten anwendet und die so erzeugte Rückgabe-Liste ermittelt.	5			
2	erläutert die Funktionsweise der Methode.	3			
3	erläutert im Sachkontext, welche Information die Methode liefert.	4			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe c)	**12**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine Strategie für einen Algorithmus der Methode.	5			
2	implementiert die Methode entsprechend der Dokumentation.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (11) …… ……					
	Summe Teilaufgabe d)	**11**			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	erläutert, welche Veränderungen in der Modellierung vorgenommen werden müssen.	3			
2	begründet, warum der Vorschlag problematisch ist, wenn verschiedene Scooter die gleiche Entfernung zur Firmenzentrale haben.	3			
3	beurteilt mit Bezug auf die beiden Anwendungsbeispiele, inwiefern der Modellierungsvorschlag des Praktikanten sinnvoll ist.	6			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …… ……					
	Summe Teilaufgabe e)	**12**			

	Summe insgesamt	**50**			

Festlegung der Gesamtnote (Bitte nur bei der letzten bearbeiteten Aufgabe ausfüllen.)

	Lösungsqualität			
	maximal erreichbare Punktzahl	**EK**	**ZK**	**DK**
Übertrag der Punktsumme aus der ersten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der zweiten bearbeiteten Aufgabe	**50**			
Übertrag der Punktsumme aus der dritten bearbeiteten Aufgabe	**50**			
Punktzahl der gesamten Prüfungsleistung	**150**			
aus der Punktsumme resultierende Note gemäß nachfolgender Tabelle				
Note ggf. unter Absenkung um bis zu zwei Notenpunkte gemäß § 13 Abs. 2 APO-GOSt				
Paraphe				

Berechnung der Endnote nach Anlage 4 der Abiturverfügung auf der Grundlage von § 34 APO-GOSt

Die Klausur wird abschließend mit der Note _____ (____ Punkte) bewertet.

Unterschrift, Datum:

Grundsätze für die Bewertung (Notenfindung)

Für die Zuordnung der Notenstufen zu den Punktzahlen ist folgende Tabelle zu verwenden:

Note	**Punkte**	**Erreichte Punktzahl**
sehr gut plus	15	150 – 143
sehr gut	14	142 – 135
sehr gut minus	13	134 – 128
gut plus	12	127 – 120
gut	11	119 – 113
gut minus	10	112 – 105
befriedigend plus	9	104 – 98
befriedigend	8	97 – 90
befriedigend minus	7	89 – 83
ausreichend plus	6	82 – 75
ausreichend	5	74 – 68
ausreichend minus	4	67 – 60
mangelhaft plus	3	59 – 50
mangelhaft	2	49 – 41
mangelhaft minus	1	40 – 30
ungenügend	0	29 – 0