Name: _____

Abiturprüfung 2018

Informatik, Leistungskurs

Aufgabenstellung:

Um Einbrüche besser verhindern zu können, möchte die Polizei prognostizieren, wo demnächst eingebrochen wird. Die Erfahrung zeigt, dass Einbrüche häufig in der Nähe bereits begangener Einbrüche stattfinden.

Die Polizei hat begonnen, erste Informationen über Einbrüche zu verwalten. Die Einbruchgefahr wird für jedes Grundstück über einen sogenannten Gefährdungsindex bestimmt. Je größer dieser ist, desto höher ist die Einbruchgefahr.

Es wurde ein Einbruchfrühwarnsystem zur Erprobung entwickelt.

Abbildung 1 zeigt einen Ausschnitt des Implementationsdiagramms des Informatiksystems. Eine Dokumentation relevanter Klassen finden Sie im Anhang.

Name: _____

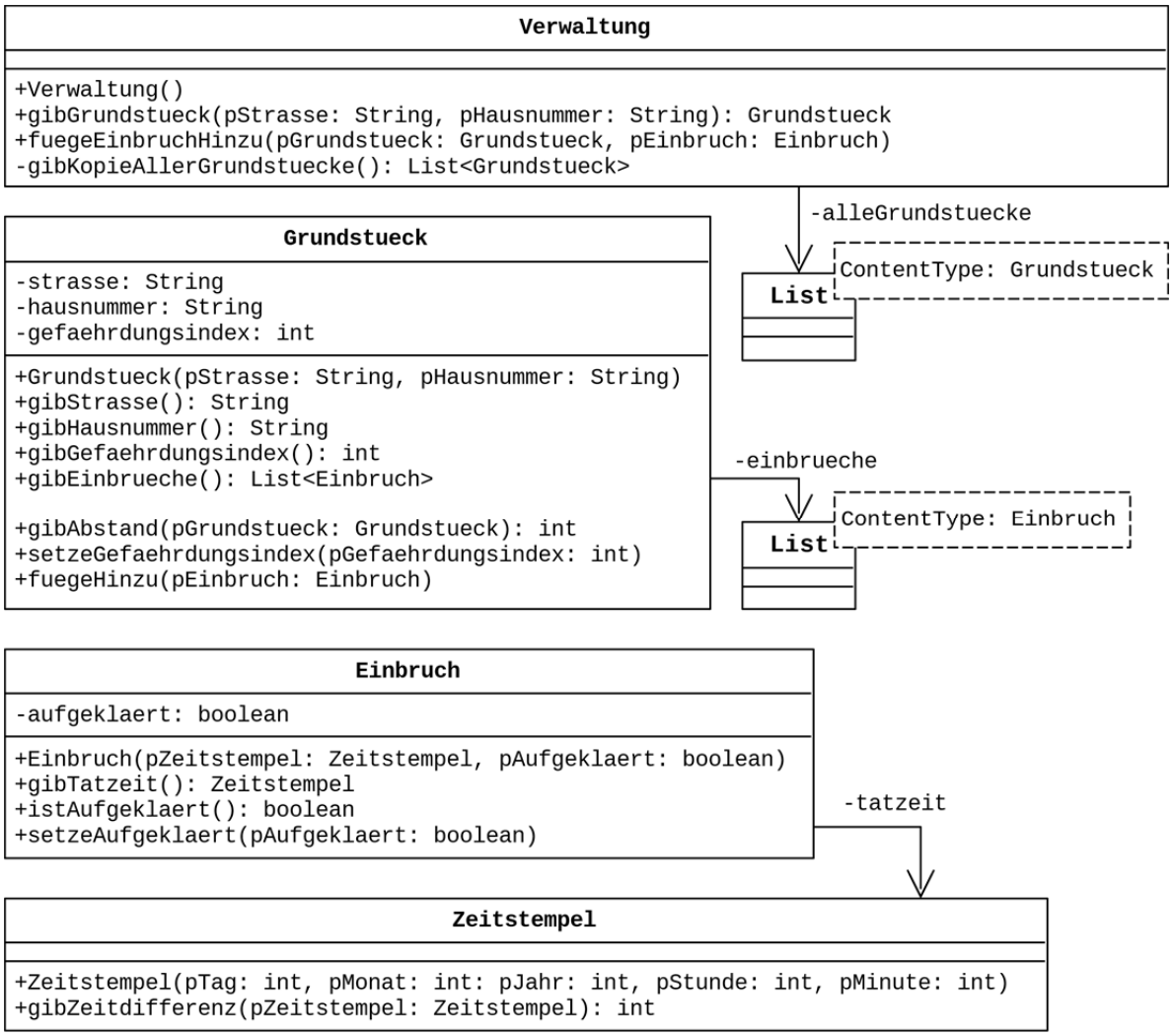

Abbildung 1: Ausschnitt aus dem Implementationsdiagramm

a) *Beschreiben Sie die Beziehungen zwischen den Klassen* `Verwaltung`, `Grundstueck`, `Einbruch` *und* `Zeitstempel`.

Erläutern Sie, warum es sinnvoll ist, die Einbrüche in einer Liste zu verwalten und nicht in einem Feld (Array).

(8 Punkte)

Name: _____

b) In der Klasse `Verwaltung` wird eine Methode `ermittleGrundstueckeMitEinbruechen` benötigt, die wie folgt dokumentiert ist:

```
public List<Grundstueck> ermittleGrundstueckeMitEinbruechen(
                    Zeitstempel pTatzeit, int pMaxMinutenDifferenz)
```

Die Methode liefert eine Liste mit den Grundstücken zurück, bei denen es mindestens einen Einbruch gibt, welcher in dem Zeitfenster `pMaxMinutenDifferenz` vor der übergebenen Tatzeit bis `pMaxMinutenDifferenz` nach der übergebenen Tatzeit liegt.

Hinweis: Sie können davon ausgehen, dass der Parameter `pMaxMinutenDifferenz` nicht negativ ist. Es darf kein Grundstück mehrfach in der Ausgabeliste vorkommen.

Entwickeln Sie eine Strategie für die Arbeitsweise der Methode `ermittleGrundstueckeMitEinbruechen` *und stellen Sie diese in geeigneter Form dar.*

Implementieren Sie die Methode `ermittleGrundstueckeMitEinbruechen`.

(17 Punkte)

Name: _____

c) Zur Einbruchsprävention wurde folgende Methode für die Klasse `Verwaltung` entwickelt:

```
 1 public List<Grundstueck> gibEtwas(int pZahl) {
 2   return gibEtwas(pZahl, gibKopieAllerGrundstuecke());
 3 }

 4 private List<Grundstueck> gibEtwas(int pZahl,
                                      List<Grundstueck> pGrundstuecke) {
 5   List<Grundstueck> neueListe = new List<Grundstueck>();
 6   if (pZahl == 0 || pGrundstuecke.isEmpty()) {
 7     return neueListe;
 8   } else {
 9     pGrundstuecke.toFirst();
10     Grundstueck groesstes = null;
11     while (pGrundstuecke.hasAccess()) {
12       Grundstueck aktuelles = pGrundstuecke.getContent();
13       if (groesstes == null || aktuelles.gibGefaehrdungsindex()
                                > groesstes.gibGefaehrdungsindex()) {
14         groesstes = aktuelles;
15       }
16       pGrundstuecke.next();
17     }
18     neueListe.append(groesstes);
19     pGrundstuecke.toFirst();
20     while (pGrundstuecke.hasAccess()) {
21       Grundstueck aktuelles = pGrundstuecke.getContent();
22       if (aktuelles == groesstes) {
23         pGrundstuecke.remove();
24       }
25       pGrundstuecke.next();
26     }
27     neueListe.concat(gibEtwas(pZahl - 1, pGrundstuecke));
28     return neueListe;
30   }
31 }
```

Name: _____

Gegeben ist die Liste `alleGrundstuecke` mit folgenden Testdaten:

Grundstück	Gefährdungsindex
`Antonstr. 1`	`1500`
`Bertastr. 2`	`1000`
`Cäsarstr. 3`	`3500`
`Dorastr. 4`	`2000`
`Emilstr. 5`	`1000`

Tabelle 1: Testdaten für die Liste `alleGrundstuecke` mit Angabe der Gefährdungsindizes

Analysieren Sie die Methoden `gibEtwas`*, indem Sie jeweils die Rückgabe der Methodenaufrufe* `gibEtwas(3)` *und* `gibEtwas(4)` *für die Testdaten aus Tabelle 1 angeben.*

Erläutern Sie die Funktionsweise und Funktionalität der Methoden im Sachkontext.

Erläutern Sie, welches Problem entsteht, wenn man statt der Kopie der Grundstücksliste (vgl. Zeile 2) die Originalgrundstücksliste `alleGrundstuecke` *verwenden würde.*

(17 Punkte)

d) Die Polizei möchte das Frühwarnsystem optimieren, weil sie festgestellt hat, dass die Gefährdungsindizes besonders häufig abgefragt und selten neue Einbrüche hinzugefügt werden.

Bei dem bisherigen Modell wird der Gefährdungsindex beim Einfügen eines neuen Einbruchs für alle Grundstücke neu berechnet. Die Abfrage des Gefährdungsindexes mithilfe der Methode `gibGefaehrdungsindex` für ein Objekt der Klasse `Grundstueck` ist ohne weiteren Berechnungsaufwand möglich.

Von einem Softwareentwickler wird vorgeschlagen, auf das Attribut `gefaehrdungsindex` in der Klasse `Grundstueck` zu verzichten. Man könne schließlich den Gefährdungsindex in der Methode `gibGefaehrdungsindex` der Klasse `Grundstueck` auch berechnen, wenn jedes Grundstück zusätzlich all seine Grundstücke im betreffenden Umkreis kennt. Der Softwareentwickler behauptet, dass sein Vorschlag in jedem Fall besser sei.

Beurteilen Sie den Vorschlag des Softwareentwicklers unter Berücksichtigung der Laufzeit- und Speichereffizienz.

(8 Punkte)

Zugelassene Hilfsmittel:

- Taschenrechner (graphikfähiger Taschenrechner / CAS-Taschenrechner)
- Wörterbuch zur deutschen Rechtschreibung

Name: _____

Anhang

Dokumentationen der verwendeten Klassen

Die Klasse Verwaltung

Ein Objekt der Klasse `Verwaltung` dient zur Verwaltung der Grundstücke und Einbrüche.

Ausschnitt aus der Dokumentation der Klasse `Verwaltung`

Konstruktor **`Verwaltung()`**

Ein neues Verwaltungsobjekt wird erzeugt.

Anfrage **`Grundstueck gibGrundstueck(String pStrasse, String pHausnummer)`**

Die Methode liefert das Grundstücksobjekt mit der Adresse (`pStrasse`, `pHausnummer`) zurück. Falls zur Adresse kein Grundstücksobjekt existiert, wird `null` zurückgeliefert.

Auftrag **`void fuegeEinbruchHinzu(Grundstueck pGrundstueck, Einbruch pEinbruch)`**

Dem Grundstück `pGrundstueck` wird der Einbruch `pEinbruch` hinzugefügt.

Die Klasse verfügt außerdem über folgende private Methode:

Anfrage **`List<Grundstueck> gibKopieAllerGrundstuecke()`**

Die Methode liefert eine tiefe Kopie, d. h. eine neue Liste, deren Einträge jeweils vollständige Kopien aller Grundstücksobjekte enthalten, zurück.
Hinweis: Die zurückgelieferte Liste referenziert nicht die Originaldaten, sondern inhaltsgleiche und damit unabhängige Kopien.

Name: _____

Die Klasse Grundstueck

Ein Objekt der Klasse `Grundstueck` speichert die Informationen zur Straße, zur Hausnummer, zum Gefährdungsindex und die Einbrüche.

Ausschnitt aus der Dokumentation der Klasse `Grundstueck`

Konstruktor **`Grundstueck(String pStrasse, String pHausnummer)`**

Ein neues Grundstücksobjekt wird mit den Informationen `pStrasse` und `pHausnummer` erzeugt. Der Gefährdungsindex wird mit dem Wert `0` initialisiert. Die Liste der Einbrüche ist leer.

Anfrage **`String gibStrasse()`**

Die Straße wird zurückgegeben.

Anfrage **`String gibHausnummer()`**

Die Hausnummer wird zurückgegeben.

Anfrage **`int gibGefaehrdungsindex()`**

Der Gefährdungsindex wird zurückgegeben.

Anfrage **`List<Einbruch> gibEinbrueche()`**

Die Liste mit den Einbruchsobjekten wird zurückgeliefert.

Anfrage **`int gibAbstand(Grundstueck pGrundstueck)`**

Der Abstand zu dem im Parameter übergebenen Grundstück `pGrundstueck` wird (auf eine nicht näher spezifizierte Weise) in Metern zurückgegeben.

Auftrag **`void setzeGefaehrdungsindex(int pGefaehrdungsindex)`**
Dem Gefährdungsindex wird der im Parameter übergebene Wert `pGefaehrdungsindex` zugewiesen.

Auftrag **`void fuegeHinzu(Einbruch pEinbruch)`**
Der Einbruch `pEinbruch` wird der Einbruchsliste des Grundstücks hinzugefügt.

Name: _____

Die Klasse Einbruch

Ein Objekt der Klasse `Einbruch` verwaltet die Tatzeit als Zeitstempel und die Information, ob der Einbruch aufgeklärt ist.

Ausschnitt aus der Dokumentation der Klasse Einbruch

Konstruktor **`Einbruch(Zeitstempel pZeitstempel, boolean pAufgeklaert)`**
Ein Einbruchsobjekt mit dem übergebenen `Zeitstempel` und dem als Parameter übergebenen Wert, ob der Einbruch aufgeklärt ist, wird erzeugt.

Anfrage **`Zeitstempel gibTatzeit()`**
Liefert die Tatzeit als `Zeitstempel` zurück.

Anfrage **`boolean istAufgeklaert()`**
Liefert den Attributwert zurück, ob der Einbruch aufgeklärt ist. Wenn der Einbruch aufgeklärt ist, dann wird `true` zurückgeliefert und sonst `false`.

Auftrag **`void setzeAufgeklaert(boolean pAufgeklaert)`**
Der Attributwert, ob der Einbruch aufgeklärt ist, wird auf `pAufgeklaert` gesetzt.

Die Klasse Zeitstempel

Ein Objekt der Klasse `Zeitstempel` dient zur Repräsentation eines Zeitpunktes.

Ausschnitt aus der Dokumentation der Klasse `Zeitstempel`

Konstruktor **`Zeitstempel(int pTag, int pMonat, int pJahr, int pStunde, int pMinute)`**
Ein Zeitstempelobjekt mit den im Parameter übergebenen Werten für den Tag, den Monat, das Jahr, die Stunde und die Minute wird erzeugt.

Anfrage **`int gibZeitdifferenz(Zeitstempel pZeitstempel)`**
Die Zeitdifferenz in Minuten zu dem im Parameter übergebenen Zeitstempel wird zurückgegeben.
Hinweis: Der zurückgegebene Wert ist negativ, wenn der im Parameter übergebene Zeitstempel zeitlich vor dem aktuellen Zeitstempel liegt.
Der zurückgegebene Wert ist positiv, wenn der im Parameter übergebene Zeitstempel zeitlich später als der aktuelle Zeitstempel liegt.

Name: _____

Die generische Klasse List<ContentType>

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 können 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>`

Konstruktor **`List()`**

Eine leere Liste wird erzeugt. Objekte, die in dieser Liste verwaltet werden, müssen vom Typ `ContentType` sein.

Anfrage **`boolean isEmpty()`**

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

Anfrage **`boolean hasAccess()`**

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

Auftrag **`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`.

Auftrag **`void toFirst()`**

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

Auftrag **`void toLast()`**

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

Name: _____

Anfrage **`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.

Auftrag **`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.

Auftrag **`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.

Auftrag **`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.

Auftrag **`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.

Auftrag **`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.

Unterlagen für die Lehrkraft

Abiturprüfung 2018

Informatik, Leistungskurs

1. Aufgabenart

Modellierung, Implementation und Analyse kontextbezogener Problemstellungen mit Schwerpunkt auf den Inhaltsfeldern Daten und ihre Strukturierung, Algorithmen und Informatiksysteme

2. Aufgabenstellung[1]

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zum Kernlehrplan und zu den Vorgaben 2018

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
       Array bis zweidimensional
       Lineare Liste

   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

- Taschenrechner (graphikfähiger Taschenrechner / CAS-Taschenrechner)
- 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)

Ein Objekt der Klasse `Verwaltung` verwaltet in einer linearen Liste Objekte vom Typ `Grundstueck`.

Ein Objekt der Klasse `Grundstueck` verwaltet in einer linearen Liste Objekte vom Typ `Einbruch`.

Ein Objekt der Klasse `Einbruch` verwaltet ein Objekt vom Typ `Zeitstempel`.

Die (maximale) Anzahl der Einbrüche für jedes Grundstück ist im Vorfeld nicht bekannt. Da die Arraygröße im Nachhinein nicht ohne Zusatzaufwand verändert werden kann, ist die Verwaltung der Einbrüche in einem Array ungünstig.

Teilaufgabe b)

Strategie:

- Erzeuge eine anfangs leere Liste für die auszugebenden Grundstücke.
- Durchlaufe die Liste mit allen Grundstücken.
  - Durchlaufe pro Grundstück die jeweilige Einbruchsliste.
    Wenn ein Grundstück mit Einbruch im gesuchten Zeitfenster gefunden wird, dann wird das entsprechende Grundstück an die Ausgabeliste angehängt und die Schleife über die Einbruchsliste bricht ab, um ein doppeltes Einfügen eines Grundstücks zu vermeiden.
- Rückgabe der Ausgabeliste mit den gesuchten Grundstücken.

Implementierung:

```
public List<Grundstueck> ermittleGrundstueckeMitEinbruechen(
                  Zeitstempel pTatzeit, int pMaxMinutenDifferenz) {
  List<Grundstueck> ausgabeliste = new List<Grundstueck>();
  alleGrundstuecke.toFirst();
  while (alleGrundstuecke.hasAccess()) {
    Grundstueck aktuellesGrundstueck =
                                    alleGrundstuecke.getContent();
    List<Einbruch> einbruchsliste =
                              aktuellesGrundstueck.gibEinbrueche();
    einbruchsliste.toFirst();
    boolean schonEingefuegt = false;
    while (einbruchsliste.hasAccess() && !schonEingefuegt) {
      Einbruch aktuellerEinbruch = einbruchsliste.getContent();
      int zeitdifferenz =
         aktuellerEinbruch.gibTatzeit().gibZeitdifferenz(pTatzeit);
      if (zeitdifferenz >= -pMaxMinutenDifferenz &&
                           zeitdifferenz <= pMaxMinutenDifferenz) {
        ausgabeliste.append(aktuellesGrundstueck);
        schonEingefuegt = true;
      }
      einbruchsliste.next();
    }
    alleGrundstuecke.next();
  }
  return ausgabeliste;
}
```

Teilaufgabe c)

Der Methodenaufruf `gibEtwas(3)` mit den Testdaten liefert eine Liste mit den Grundstücken `[Cäsarstr. 3, Dorastr. 4, Antonstr. 1]` zurück.

Der Methodenaufruf `gibEtwas(4)` mit den Testdaten liefert eine Liste mit den Grundstücken `[Cäsarstr. 3, Dorastr. 4, Antonstr. 1, Bertastr. 2]` zurück.

Es wird die private Methode `gibEtwas` mit dem Parameter `pZahl` und mit einer Kopie der Liste mit allen Grundstücken aufgerufen (vgl. Zeile 2).
In der privaten Methode wird zunächst eine leere Liste `neueListe` initialisiert (vgl. Zeile 5).
Wenn die im Parameter übergebene `pZahl` gleich 0 ist oder die Liste `pGrundstuecke` leer ist, dann wird die neue leere Liste zurückgegeben (Rekursionsanker). Sonst wird zunächst die Liste mit allen Grundstücken durchlaufen und das Grundstück mit dem größten Gefährdungsindex gesucht und in der lokalen Variable `groesstes` gespeichert (vgl. Zeilen 9 – 17), an die `neueListe` angehängt (vgl. Zeile 18) und aus der Liste mit den Kopien aller Grundstücke `pGrundstuecke` entfernt (vgl. Zeilen 19 – 26).
An die `neueListe` wird das Ergebnis des rekursiven Methodenaufrufs `gibEtwas(pZahl – 1, pGrundstuecke)` angehängt.
Den Rückgabewert der Methode bildet die (ggf. leere) Liste `neueListe`.

Die rekursive Methode `gibEtwas` liefert die als Parameter angegebene Anzahl der gefährdetsten Grundstücke in einer neuen Liste zurück. Die Liste ist absteigend nach Gefährdungsindex sortiert. Wenn der Fall eintritt, dass mehrere Grundstücke den gleichen Gefährdungsindex haben, aber nicht alle diese Grundstücke aufgrund des Parameters `pZahl` ausgegeben werden, entscheidet die Reihenfolge in der Liste, welche Grundstücke ausgegeben werden.

Wenn man in dieser Methode die Originalgrundstücksliste `alleGrundstuecke` verwendet, werden die `pZahl` gefährdetsten Grundstücke aus der Originalgrundstücksliste entfernt (vgl. Zeile 23). Auf diese Grundstücke kann dann nicht mehr zugegriffen werden.

Teilaufgabe d)

Der Softwareentwickler möchte den Speicherbedarf minimieren, indem er auf ein Integerattribut verzichtet. Allerdings muss bei seinem Vorschlag jedes Grundstück eine Datensammlung von Grundstücksobjekten im betreffenden Umkreis verwalten.
Unter Effizienz wird allerdings nicht nur die Minimierung des Speicherbedarfs, sondern auch die Laufzeitoptimierung (Zahl der Operationen) verstanden.

Da die Gefährdungsindizes laut Aufgabenstellung besonders häufig abgefragt und Einbrüche selten eingefügt werden, sind die Laufzeitüberlegungen zur Abfrage des Gefährdungsindexes stärker zu gewichten als die zum Einfügen eines Einbruchs.

Laufzeitüberlegungen zur Abfrage des Gefährdungsindexes:
Beim Vorschlag des Softwareentwicklers muss bei der Abfrage des Gefährdungsindexes jener immer komplett berechnet werden. Dazu müssen bei einer Abfrage für einen Gefährdungsindex die Liste der Grundstücke im betreffenden Umkreis und zusätzlich die jeweiligen Einbruchslisten durchlaufen werden.

Laufzeitüberlegungen zum Einfügen eines Einbruchs:
Bei dem Vorschlag des Softwareentwicklers entfällt beim Einfügen eines neuen Einbruchs der Zeitaufwand für die Neuberechnung aller Gefährdungsindizes für alle Grundstücke. Im ursprünglichen Modell besteht der Aufwand darin, dass der Gefährdungsindex bei den Grundstücken im Umkreis angepasst werden muss, d. h., die Liste mit allen Grundstücken muss einmal durchlaufen werden.

Durch die Investition in das Attribut `gefaehrdungsindex` in der Klasse `Grundstueck` kann der Aufwand für die Berechnung des Gefährdungsindexes auf den pro Einbruch einmaligen Vorgang des Einfügens reduziert werden.

Bei der Abwägung zwischen der Minimierung des Speicherbedarfs und der Laufzeitoptimierung ist in diesem Fall der Vorschlag des Softwareentwicklers abzulehnen.

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	beschreibt die Beziehungen zwischen den Klassen.	5			
2	erläutert, warum es sinnvoll ist, die Einbrüche in einer Liste zu verwalten und nicht in einem Feld (Array).	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe a)	**8**			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	entwickelt eine Strategie für die Arbeitsweise der Methode und stellt diese in geeigneter Form dar.	8			
2	implementiert die Methode.	9			
Sachlich richtige Lösungsalternative zur Modelllösung: (17) …… ……					
	Summe Teilaufgabe b)	**17**			

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

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	analysiert die Methode und gibt die Rückgabe der Methodenaufrufe für die Testdaten an.	6			
2	erläutert die Funktionsweise und Funktionalität der Methoden im Sachkontext.	8			
3	erläutert, welches Problem entsteht, wenn man statt der Kopie der Grundstücksliste die Originalgrundstücksliste verwenden würde.	3			
Sachlich richtige Lösungsalternative zur Modelllösung: (17) …… ……					
	Summe Teilaufgabe c)	**17**			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl**	**EK**	**ZK**	**DK**
1	beurteilt den Vorschlag des Softwareentwicklers unter Berücksichtigung der Laufzeit- und Speichereffizienz.	8			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …… ……					
	Summe Teilaufgabe d)	**8**			

	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