Name: _____

Beispielaufgabe

Informatik, Grundkurs

Aufgabenstellung:

In mehreren Windparks wird mit Hilfe von Windrädern Strom erzeugt. Diese Windparks wurden bzw. werden in einem bestimmten Jahr in Betrieb genommen. Zusätzlich wird der Status protokolliert, d. h. ob ein Windpark geplant, bereits im Bau, im Teilbetrieb oder im Vollbetrieb ist.

In einem Windpark werden verschieden viele Windräder eines oder mehrerer Typen aufgestellt. Die Windradtypen können von unterschiedlichen Herstellern produziert sein.
Die maximal mögliche Leistung eines Windradtyps misst sich in Megawatt. Über Hersteller ist bekannt, an welchem Ort sie ihren Hauptsitz haben.

Das Unternehmen wählt für den Erstansatz eine Datenbankmodellierung, die im folgenden Entity-Relationship-Diagramm dargestellt ist:

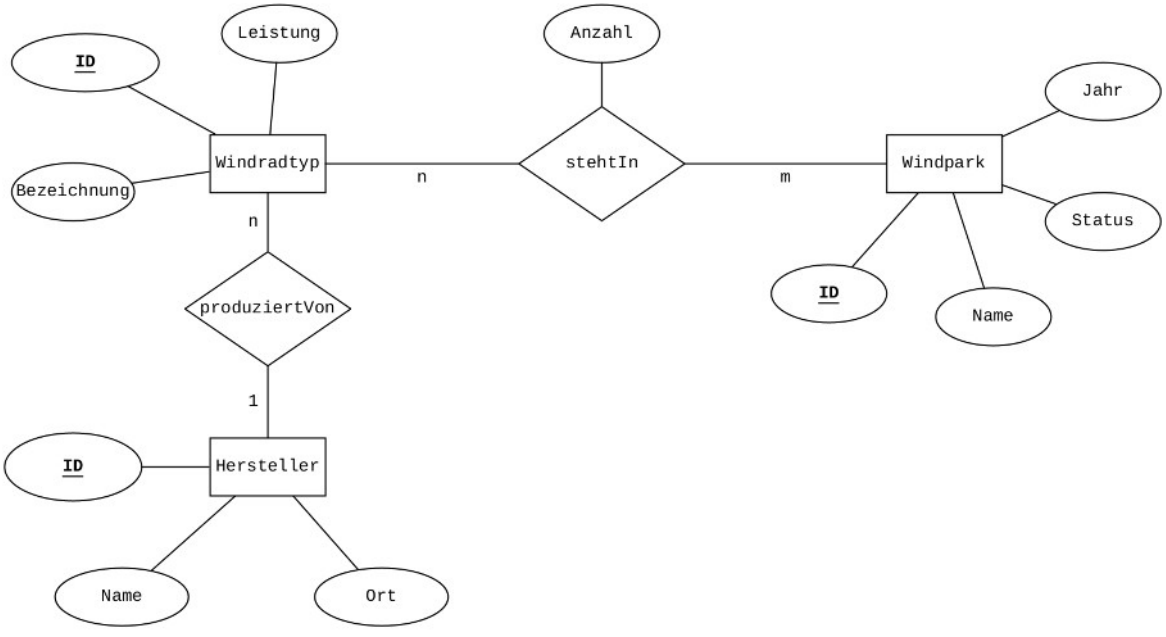

Abbildung 1: Datenbankmodellierung zur Windparkverwaltung

Name: _____

Diese Modellierung wird durch das folgende in der zweiten Normalform vorliegende relationale Datenbankschema realisiert:

Windradtyp(ID, Bezeichnung, Leistung, ↑HerstellerID)

stehtIn(↑WindradtypID, ↑WindparkID, Anzahl)

Windpark(ID, Name, Jahr, Status)

Hersteller(ID, Name, Ort)

Abbildung 2: Datenbankschema zur Windparkverwaltung

a) *Ermitteln Sie mit Hilfe des Auszugs aus der Beispieldatenbank im Anhang, an welchem Ort alle Windräder des Windparks* `Wikinger` *produziert wurden bzw. werden.*

Erläutern Sie die gewählten Kardinalitäten der beiden Beziehungstypen im Entity-Relationship-Diagramm in Abbildung 1 und erläutern Sie deren Umsetzung im gegebenen relationalen Datenbankschema.

(2 + 4 + 4 Punkte)

Als zusätzliche Anforderung sollen weitere technische Daten einzelner Windräder wie Flügellänge und Masthöhe in der Datenbank verwaltet werden. Windräder mit identischer Bezeichnung haben gleichlange Flügel und gleichhohe Masten.

Ein Datenbankentwickler schlägt vor, das Relationenschema `stehtIn` wie folgt zu erweitern:

stehtIn(↑WindradtypID, ↑WindparkID, Anzahl, Fluegellaenge,
Masthoehe)

b) *Begründen Sie, dass das Relationenschema* `stehtIn` *mit dieser Erweiterung nicht mehr in zweiter Normalform vorliegt.*

Modifizieren Sie das in Abbildung 2 gegebene Datenbankschema so, dass Flügellänge und Masthöhe gespeichert werden können und das Datenbankschema weiterhin in zweiter Normalform vorliegt.

Erläutern Sie Ihre Veränderungen am Datenbankschema aus Abbildung 2.

(3 + 3 + 3 Punkte)

Name: _____

Im Folgenden ist eine SQL-Abfrage zu den Windparks gegeben.

```
1  SELECT Windpark.ID, Windpark.Name,
2         SUM(Windradtyp.Leistung * stehtIn.Anzahl) AS Gesamt
3  FROM Windpark
4    INNER JOIN stehtIn
5      ON Windpark.ID = stehtIn.WindparkID
6    INNER JOIN Windradtyp
7      ON stehtIn.WindradtypID = Windradtyp.ID
8  GROUP BY Windpark.ID
9  ORDER BY Gesamt ASC
```

Die Abfrage liefert folgendes Ergebnis:

Windpark.ID	Windpark.Name	Gesamt
1	Alpha ventus	60.5
6	Nordergründe	110.7
3	Gode Wind 2	252
9	Sandbank	288
4	Gode Wind 1	330
7	Nordsee One	332.1
5	Wikinger	353.5
10	Arkona	360
11	Merkur Offshore	396
8	Veja Mate	402

Abbildung 3: Ergebnis der SQL-Abfrage

c) *Analysieren und erläutern Sie die SQL-Abfrage.*

Erläutern Sie, welches Ergebnis die SQL-Abfrage ohne den Zusatz `GROUP BY Windpark.ID` *ermitteln würde.*

Erläutern Sie im Sachkontext, welche Auswirkungen es hätte, wenn in der SQL-Abfrage in den Zeilen 4 und 7 mit `LEFT JOIN` *anstelle von* `INNER JOIN` *gearbeitet werden würde.*

(4 + 3 + 3 Punkte)

Name: _____

Ein Programm soll nun die Verwaltung der Windparks ermöglichen. Ein Ausschnitt der Modellierung dieses Programms ist dem folgenden Implementationsdiagramm zu entnehmen.

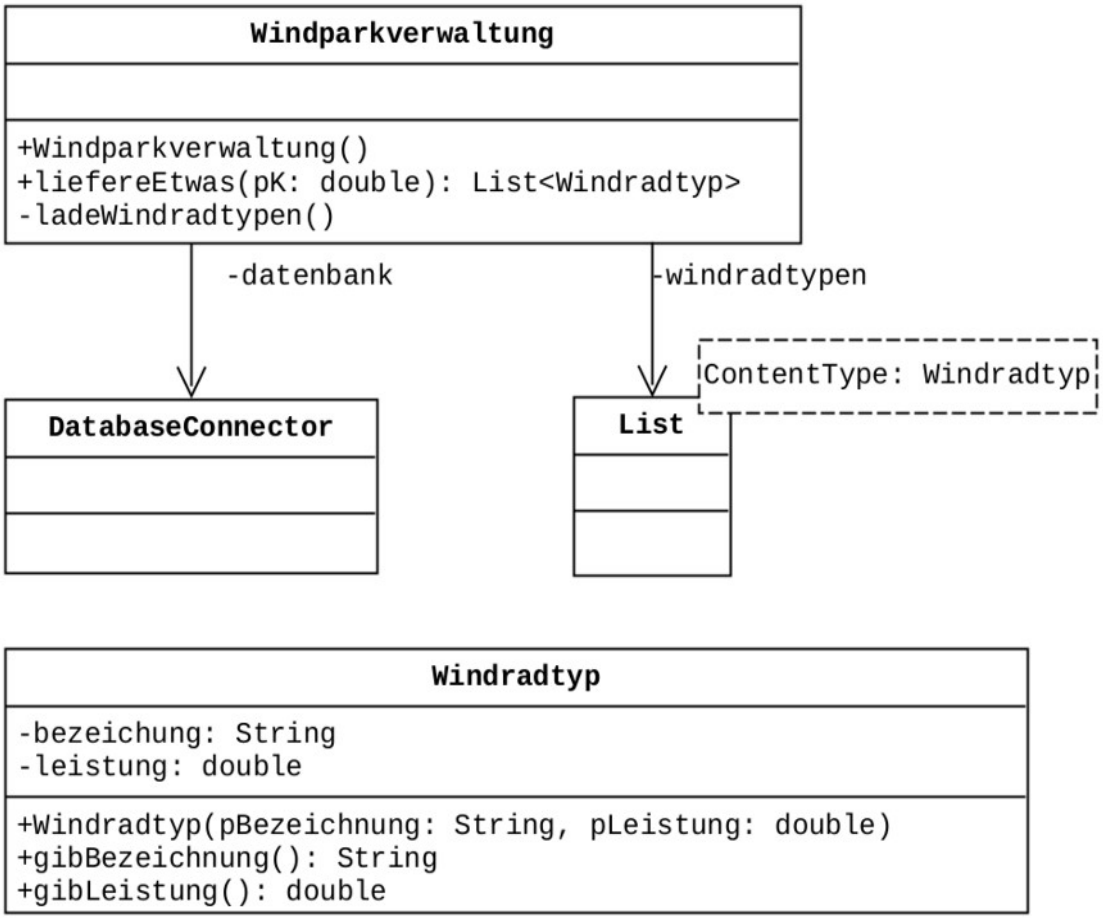

Abbildung 4: Ausschnitt aus dem Implementationsdiagramm

Name: _____

d) Der Konstruktor der Klasse `Windparkverwaltung` erstellt das Objekt `datenbank` vom Typ `DatabaseConnector` und stellt damit die Verbindung zur Datenbank her.

Die Methode `ladeWindradtypen` der Klasse `Windparkverwaltung` soll die Windradtypen laden und entsprechende Objekte vom Typ `Windradtyp` in die neu zu erstellende Liste `windradtypen` eintragen.

Von der Methode `ladeWindradtypen` ist folgender Methodenkopf und ein Teil des Methodenrumpfes gegeben:

```
1  private void ladeWindradtypen() {
2   String sql =
3    " SELECT Windradtyp.Bezeichnung, Windradtyp.Leistung  " +
4    " FROM Windradtyp                                     ";
5    // Hier Quelltext ergaenzen.
6  }
```

Vervollständigen Sie den Quelltext der Methode `ladeWindradtypen` *der Klasse* `Windparkverwaltung`.

(8 Punkte)

Name: _____

e) Um die Leistungsfähigkeit der Windradtypen zu beurteilen, stellt ein Entwickler des Softwareunternehmens zwei unterschiedliche Ansätze zur Verfügung.

1. Ansatz: In der Klasse `Windparkverwaltung` wird die folgende Methode `liefereEtwas` zur Auswertung bereitgestellt.

```
 1  public List<Windradtyp> liefereEtwas(double pK) {
 2    List<Windradtyp> auswahlTypen = new List<Windradtyp>();
 3    windradtypen.toFirst();
 4    while (windradtypen.hasAccess()) {
 5      Windradtyp tmpWindradtyp = windradtypen.getContent();
 6      if (tmpWindradtyp.gibLeistung() < pK) {
 7        boolean eingefuegt = false;
 8        auswahlTypen.toFirst();
 9        while (auswahlTypen.hasAccess() & !eingefuegt) {
10          if (auswahlTypen.getContent().gibLeistung() >
                                  tmpWindradtyp.gibLeistung()) {
11            auswahlTypen.insert(tmpWindradtyp);
12            eingefuegt = true;
13          } else {
14            auswahlTypen.next();
15          }
16        }
17        if (!eingefuegt) {
18          auswahlTypen.append(tmpWindradtyp);
19        }
20      }
21      windradtypen.next();
22    }
23    return auswahlTypen;
24  }
```

2. Ansatz: Folgendes SQL-Statement soll die Auswertung vornehmen:

```
SELECT Windradtyp.Bezeichnung, Windradtyp.Leistung
FROM Windradtyp
WHERE Windradtyp.Leistung >= 5.9
ORDER BY Windradtyp.Leistung, Windradtyp.Bezeichnung
```

Name: _____

Für den ersten Ansatz sei eine Liste `windradtypen` mit folgenden Beispieleinträgen (Bezeichnung und Leistung) gegeben:

SWT-6	**5M**	**SWT-4**	**AD 5**	**6.2M**
6.0	5.08	4.0	5.05	6.15

Abbildung 5: Beispielhafte Belegung der Liste `windradtypen`

Analysieren Sie die Methode `liefereEtwas`*, indem Sie die Methode auf die in Abbildung 5 gegebene Belegung der Liste mit dem Aufruf* `liefereEtwas(5.9)` *anwenden und schrittweise den Aufbau der Liste* `windradtypAuswahl` *dokumentieren.*

Erläutern Sie für die in Abbildung 5 gegebene Belegung schrittweise den Aufbau der Liste `windradtypAuswahl`.

Erläutern Sie, welche Aufgabe die Methode `liefereEtwas` *im Sachzusammenhang hat.*

Für eine Analyse des zweiten Ansatzes soll die Datenbanktabelle `Windradtyp` nur die ersten fünf Datensätze der Beispieldatenbanktabelle `Windradtyp` in der Anlage enthalten.

Beurteilen Sie, ob das SQL-Statement des zweiten Ansatzes dieselben Windradtypen ermittelt, die die Methode `liefereEtwas` *in der Liste zurückgibt.*

(3 + 3 + 3 + 4 Punkte)

Zugelassene Hilfsmittel:

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

Name: _____

Anlage: Auszug aus der Beispieldatenbank

Windradtyp			
ID	**Bezeichnung**	**Leistung**	**HerstellerID**
1	SWT-6	6	1
2	5M	5.08	3
3	SWT-4	4	1
4	AD 5	5.05	5
5	6.2M	6.15	6
6	Haliade 150	6	2
7	Multibrid M5000	5	4
8	Gamesa	11	1

stehtIn		
WindradtypID	**WindparkID**	**Anzahl**
2	1	6
7	1	6
1	3	42
1	4	55
4	5	70
5	6	18
5	7	54
1	8	67
3	9	72
1	10	60
6	11	66

Name: _____

Windpark			
ID	**Name**	**Jahr**	**Status**
1	alpha ventus	2010	Vollbetrieb
2	Gode Wind 3	2014	Planung
3	Gode Wind 2	2016	Vollbetrieb
4	Gode Wind 1	2016	Vollbetrieb
5	Wikinger	2017	Teilbetrieb
6	Nordergründe	2017	Teilbetrieb
7	Nordsee One	2017	Teilbetrieb
8	Veja Mate	2017	Vollbetrieb
9	Sandbank	2017	Vollbetrieb
10	Arkona	2018	Bau
11	Merkur Offshore	2018	Bau

Hersteller		
ID	**Name**	**Ort**
1	Siemens	Bremen
2	GeneralElectrics	Salzbergen
3	REpower	Dortmund
4	AREVA	Bremerhaven
5	Adwen	Bremerhaven
6	Senvion	Hamburg

Name: _____

Anhang:

Die Klasse `Windparkverwaltung`

Ein Objekt der Klasse **`Windparkverwaltung`** verwaltet eine Liste von Objekten der Klasse `Windradtyp`.
Beim Erzeugen des Objekts wird eine Verbindung zu einer relationalen Datenbank aufgebaut, es werden den Daten entsprechende Windradtypobjekte erstellt und in der Liste gespeichert.

Ausschnitt aus der Dokumentation der Klasse `Windparkverwaltung`

`Windparkverwaltung()`

Ein Objekt vom Typ `Windparkverwaltung` wird initialisiert, die in einer relationalen Datenbank gespeicherten Windradtypen werden ausgelesen und als eigenständige Objekte in einer Liste abgelegt.

`List<Windpark> liefereEtwas()`

Der Quelltext dieser Methode ist im Aufgabenteil e) zu analysieren.

Die Klasse `Windradtyp`

Ein Objekt der Klasse **`Windradtyp`** speichert die Informationen zur Bezeichnung und zur Leistung.

Ausschnitt aus der Dokumentation der Klasse `Windradtyp`

`Windradtyp(String pBezeichnung, double pLeistung)`

Ein neues Windradtypobjekt wird mit den Informationen `pBezeichnung` und `pLeistung` erzeugt.

`String gibBezeichnung()`

Die Bezeichnung des Windradtyps wird zurückgegeben.

`double gibLeistung()`

Die mögliche Leistung eines Windradtyps wird zurückgegeben.

Name: _____

Die Klasse `DatabaseConnector`

Ein Objekt der Klasse **`DatabaseConnector`** ermöglicht die Abfrage und Manipulation einer relationalen Datenbank.
Beim Erzeugen des Objekts wird eine Datenbankverbindung aufgebaut, so dass anschließend SQL-Anweisungen an diese Datenbank gerichtet werden können.

Dokumentation der Klasse `DatabaseConnector`

```
DatabaseConnector(String pIP, int pPort, String pDatabase,
          String pUsername, String pPassword)
```

Ein Objekt vom Typ `DatabaseConnector` wird erstellt, und eine Verbindung zur Datenbank wird aufgebaut. Mit den Parametern `pIP` und `pPort` werden die IP-Adresse und die Port-Nummer übergeben, unter denen die Datenbank mit Namen `pDatabase` zu erreichen ist. Mit den Parametern `pUsername` und `pPassword` werden Benutzername und Passwort für die Datenbank übergeben.

```
void executeStatement(String pSQLStatement)
```

Der Auftrag schickt den im Parameter `pSQLStatement` enthaltenen SQL-Befehl an die Datenbank ab.
Handelt es sich bei `pSQLStatement` um einen SQL-Befehl, der eine Ergebnismenge liefert, so kann dieses Ergebnis anschließend mit der Methode `getCurrentQueryResult` abgerufen werden.

```
QueryResult getCurrentQueryResult()
```

Die Anfrage liefert das Ergebnis des letzten mit der Methode `executeStatement` an die Datenbank geschickten SQL-Befehls als Objekt vom Typ `QueryResult` zurück.
Wurde bisher kein SQL-Befehl abgeschickt oder ergab der letzte Aufruf von `executeStatement` keine Ergebnismenge (z.B. bei einem INSERT-Befehl oder einem Syntaxfehler), so wird `null` geliefert.

```
String getErrorMessage()
```

Die Anfrage liefert `null` oder eine Fehlermeldung, die sich jeweils auf die letzte zuvor ausgeführte Datenbankoperation bezieht.

```
void close()
```

Die Datenbankverbindung wird geschlossen.

Name: _____

Die Klasse `QueryResult`

Ein Objekt der Klasse **`QueryResult`** stellt die Ergebnistabelle einer Datenbankanfrage mit Hilfe der Klasse `DatabaseConnector` dar. Objekte dieser Klasse werden nur von der Klasse `DatabaseConnector` erstellt. Die Klasse verfügt über keinen öffentlichen Konstruktor.

Dokumentation der Klasse `QueryResult`

`String[][] getData()`

Die Anfrage liefert die Einträge der Ergebnistabelle als zweidimensionales Feld vom Typ `String`. Der erste Index des Feldes stellt die Zeile und der zweite die Spalte dar (d. h. `String[zeile][spalte]`).

`String[] getColumnNames()`

Die Anfrage liefert die Bezeichner der Spalten der Ergebnistabelle als Feld vom Typ `String` zurück.

`String[] getColumnTypes()`

Die Anfrage liefert die Typenbezeichnung der Spalten der Ergebnistabelle als Feld vom Typ `String` zurück. Die Bezeichnungen entsprechen den Angaben in der Datenbank.

`int getRowCount()`

Die Anfrage liefert die Anzahl der Zeilen der Ergebnistabelle als `int`.

`int getColumnCount()`

Die Anfrage liefert die Anzahl der Spalten der Ergebnistabelle als `int`.

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 oder ein Listenobjekt an das Ende der Liste angefügt werden.

Dokumentation der Klasse `List`

`List<ContentType>()`

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.

Name: _____

`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.

`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 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` gleich `null` ist, bleibt die Liste unverändert.

`void concat(List pList)`

Die Liste `pList` wird an die Liste angehängt. Anschließend wird `pList` eine leere Liste. Das aktuelle Objekt bleibt unverändert. Falls `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.

Unterlagen für die Lehrkraft

Beispielaufgabe

Informatik, Grundkurs

1. Aufgabenart

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

2. Aufgabenstellung

siehe Prüfungsaufgabe

3. Materialgrundlage

entfällt

4. Bezüge zu dem Kernlehrplan und den Vorgaben

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
     - lineare Strukturen (Array, lineare Liste)
   - Datenbanken

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

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

2. Medien / Materialien

   entfällt

5. Zugelassene Hilfsmittel

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

6.1 Vorbemerkung zu dieser Beispielklausur

Das Ziel dieser Beispielaufgabe ist es, eine mögliche Verknüpfung der Inhaltsfelder „Daten und ihre Strukturierung“ (Objekte und Klassen, Datenbanken) und „Algorithmen“ für eine inhaltsfeldübergreifende Grundkursklausur zu zeigen.

Diese Beispielaufgabe orientiert sich an den Abiturvorgaben 2021.

Ab dem Abiturjahrgang 2021 ist für eine Grundkursklausur, die aus zwei Aufgaben besteht, eine Bearbeitungszeit von 3 Stunden und 45 Minuten vorgesehen. Diese Beispielaufgabe stellt nur eine von zwei Aufgaben dar, die zusammen eine vollständige Grundkursklausur ergeben.

6.2 Modelllösungen

Die 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)

Die Windräder des Windparks `Wikinger` wurden in Bremerhaven produziert.

Ein Windpark kann unterschiedliche Windradtypen enthalten. Windräder desselben Typs können in mehreren Windparks stehen. Also handelt es sich hierbei um einen n:m-Beziehungstyp.
Dieser Beziehungstyp muss durch ein separates Relationenschema `stehtIn` in das relationale Datenbankschema übernommen werden. In diesem Relationenschema bildet die Kombination der Fremdschlüssel `WindradtypID` und `WindparkID` den Primärschlüssel.

Ein Windradtyp wird von genau einem Hersteller produziert. Ein Hersteller kann aber auch mehrere Windradtypen produzieren. Also handelt es sich dabei um einen n:1-Beziehungstyp.
Zur Umsetzung dieses Beziehungstyps wird der Primärschlüssel des Relationenschemas `Hersteller` als Fremdschlüssel in das Schema `Windradtyp` eingebunden. So wird bei jedem Windradtyp die `HerstellerID` des Herstellers notiert.

Teilaufgabe b)

Das Attribut `ID` ist Primärschlüssel des Relationsschemas `Windradtyp`. Das Attribut `Bezeichnung` ist funktional von `ID` abhängig. Da `WindradtypID` Fremdschlüssel im Relationenschema `stehtIn` ist, ist das Attribut `Bezeichnung` auch von `WindradtypID` funktional abhängig.
Die Attribute `Fluegellaenge` und `Masthoehe` stehen in funktionaler Abhängigkeit zum Attribut `Bezeichnung`, da Windräder mit identischer Bezeichnung gleichlange Flügel und gleichhohe Masten haben. Damit sind die beiden Attribute `Fluegellaenge` und `Masthoehe` auch vom Attribut `WindradtypID` funktional abhängig.

Da das Attribut `WindradtypID` aber nur ein Teil des Primärschlüssels des Relationsschemas `stehtIn` ist, nicht aber der gesamte Primärschlüssel, sind die beiden Attribute `Fluegellaenge` und `Masthoehe` nicht voll vom Primärschlüssel funktional abhängig, was aber den Bedingungen der zweiten Normalform widerspricht.

Nimmt man die Attribute `Fluegellaenge` und `Masthoehe` in das Relationenschema `Windradtyp`, von dessen Primärschlüssel sie funktional abhängen, auf, liegt das Schema in der zweiten Normalform vor. Korrekt wäre hier also z.B. folgende Veränderung:

Windradtyp(ID, Bezeichnung, Leistung, Fluegellaenge,
Masthoehe, ↑HerstellerID)

Das Relationenschema `stehtIn` kann in der ursprünglichen Form erhalten bleiben:

stehtIn(↑WindradtypID, ↑WindparkID, Anzahl)

Teilaufgabe c)

Die SQL-Anfrage ermittelt Datensätze mit den IDs sowie den Bezeichnern der Windparks und der maximal möglichen Gesamtleistung. Dazu wird jeweils die Summe über alle Produkte aus der `Leistung` eines Windradtyps und der `Anzahl` der Windräder dieses Typs gebildet. Die Datensätze sind hierbei aufsteigend nach der maximal möglichen Gesamtleistung sortiert.
Dabei wird die Datenbanktabelle `stehtIn`, die das Attribut `Anzahl` enthält, einerseits mit der Datenbanktabelle `Windradtypen` und andererseits mit der Datenbanktabelle `Windpark` verknüpft.

Die Gruppierung bewirkt, dass die Gesamtsumme der maximal möglichen Leistung für jeden einzelnen Windpark gebildet wird. Ohne diese Gruppierung würde die Gesamtsumme über alle maximal möglichen Leistungen in allen Windparks gebildet.

Ein Austausch des ersten `INNER JOIN` gegen ein `LEFT JOIN` würde bewirken, dass alle Datensätze der linken Seite (Relation `Windpark`) mit den zugehörigen Datensätzen der rechten Seite (Datenbanktabelle `stehtIn`) verbunden werden. Ein Austausch des zweiten `INNER JOIN` verbindet dann alle diese Ergebnisdatensätze mit Hilfe der `WindradtypID` mit der Datenbanktabelle `Windradtyp`.
Dabei werden auch Datensätze der linken Seite ermittelt, für die keine Entsprechung auf der rechten Seite vorhanden ist. Es werden also alle Windparks ermittelt, also auch solche ohne zugeordnete Windradtypen, wie der sich in Planung befindende Windpark `GODE WIND 3`. Für solche Windparks wird keine Leistung angegeben.

Teilaufgabe d)

```
private void ladeWindradtypen() {
  String sql =
      " SELECT Windradtyp.Bezeichnung, Windradtyp.Leistung     " +
      " FROM Windradtyp                                        ";

  Windradtyp aktTyp;
  windradtypen = new List<Windradtyp>();
  datenbank.executeStatement(sql);
  QueryResult sqlErgebnis = datenbank.getCurrentQueryResult();
  for (int i = 0; i < sqlErgebnis.getRowCount(); i++) {
    String[] aktZeile = sqlErgebnis.getData()[i];
    aktTyp = new Windradtyp(aktZeile[0],
                                  Double.parseDouble(aktZeile[1]));
    windradtypen.append(aktTyp);
  }
}
```

Teilaufgabe e)

1. Ansatz

Schrittweiser Aufbau der Liste:

1. `auswahlTypen = {}`
2. `auswahlTypen = {}` `// 6.0 > 5.9 (SWT-6)`
3. `auswahlTypen = {5M}` `// 5.08 < 5.9`
4. `auswahlTypen = {SWT-4; 5M}` `// 4.0 < 5.9 und`
   `// 4.0 < 5.08`
5. `auswahlTypen = {SWT-4; AD 5; 5M}` `// 5.05 < 5.9 und`
   `// 5.05 > 4.0; 5.05 < 5.08`
6. `auswahlTypen = {SWT-4; AD 5; 5M}` `// 6.15 > 5.9 (6.2M)`

Die Methode `liefereEtwas` initialisiert zunächst die Liste `auswahlTypen` mit einer leeren Liste. Dann wird in der Methode die Liste aller verwalteten Windradtypen von vorne bis hinten durchlaufen: Die Leistung des ersten Windradtyps `SWT-6` liegt mit `6.0` über dem Schwellenwert von `5.9`. Also wird die Bearbeitung mit dem nächsten Windradtyp `5M` fortgesetzt. Dieser wird in die Liste `auswahlTypen` eingefügt, denn die Leistung liegt mit `5.08` unter dem Schwellenwert. Der nächste zu bearbeitende Windradtyp `SWT-4` wird mit einer Leistung von `4.0` in die Liste eingefügt und zwar vor den Windradtyp `5M`, da die Leistung geringer ist. Der dann folgenden Windradtyp `AD 5` hat eine Leistung von `5.05`. Dieser Wert liegt unter dem Schwellenwert, weswegen `AD 5` ebenfalls in die Liste `auswahlTypen` eingefügt werden muss. Die Leistung des Windradtyps `AD 5` ist kleiner als die Leistung von `5M`, aber größer als die Leistung von `SWT-4`, also wird der Windradtyp `AD 5` genau in die Mitte der Liste `auswahlTypen` eingefügt. Der letzte vorhandene Windradtyp `6.2M` hat mit `6.15` eine Leistung, die über dem Schwellenwert liegt und wird deshalb nicht in die Liste eingefügt.

Die Methode `liefereEtwas` ermittelt die Windradtypen, deren maximale Leistung unter dem als Parameter übergebenen Schwellenwert liegen und gibt diese sortiert nach der maximalen Leistung in einer linearen Liste zurück.

2. Ansatz

Das gegebene SQL-Statement hat bei der Selektion fast denselben strukturellen Aufbau wie der Quelltext der Java-Methode (Zeile 6). Mit Hilfe der WHERE-Klausel werden allerdings alle Windradtypen aus der Datenbanktabelle selektiert, deren maximale Leistung gleich oder größer als der Schwellenwert `5.9` (Megawatt) sind. Zusätzlich werden die Ergebnisdatensätze aufsteigend nach der Leistung sortiert.

Damit unterscheidet sich das Ergebnis der SQL-Abfrage von dem Ergebnis der Methode des ersten Ansatzes im Wesentlichen dadurch, dass alle Windradtypen, deren maximale Leistung gleich oder über dem Schwellenwert ermittelt werden, wohingegen die Methode diejenigen Windradtypen zurückgibt, deren maximale Leistung echt darunter liegen. Also werden nicht dieselben Windradtypen ermittelt, es sei denn, die Datenbank enthält überhaupt keine Windradtypen (in diesem Fall sind beide Ergebnisse die leere Menge, mithin also gleich).

Ein weiterer Unterschied besteht darin, dass die SQL-Anfrage Windradtypen mit identischer Leistung aufsteigend sortiert nach der Bezeichnung ausgibt.

7. Teilleistungen – Kriterien / Bewertungsbogen zur Prüfungsarbeit

Name des Prüflings: _____ Kursbezeichnung: _____

Schule: _____

Teilaufgabe a)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl (AFB)**	**EK[1]**	**ZK**	**DK**
1	ermittelt den Produktionsort der Windräder des Windparks `Wikinger`.	2 (II)			
2	erläutert die gewählten Kardinalitäten.	4 (I)			
3	erläutert die Umsetzung der Kardinalitäten im Relationenschema.	4 (II)			
Sachlich richtige Lösungsalternative zur Modelllösung: (10) …….. ……..					
	Summe Teilaufgabe a)	10			

Teilaufgabe b)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl (AFB)**	**EK**	**ZK**	**DK**
1	begründet, dass das Relationenschema mit dieser Erweiterung nicht mehr in 2. NF vorliegt.	3 (II)			
2	modifiziert das Ausgangsdatenbankschema so, dass die Anforderungen erfüllt werden und das Datenbankschema weiterhin in 2. NF vorliegt.	3 (II)			
3	erläutert die vorgenommenen Veränderungen am Datenbankschema.	3 (II)			
Sachlich richtige Lösungsalternative zur Modelllösung: (7) …….. ……..					
	Summe Teilaufgabe b)	9			

1 EK = Erstkorrektur; ZK = Zweitkorrektur; DK = Drittkorrektur

Teilaufgabe c)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl (AFB)**	**EK**	**ZK**	**DK**
1	analysiert und erläutert die SQL-Abfrage.	4 (II)			
2	Erläutert, welches Ergebnis die SQL-Abfrage ohne des Zusatz `GROUP BY` ermitteln würde.	3 (II)			
3	erläutert im Sachkontext die Auswirkungen eines Austausches des `INNER JOIN` gegen ein `LEFT JOIN`.	3 (II)			
Sachlich richtige Lösungsalternative zur Modelllösung: (12) …….. ……..					
	Summe Teilaufgabe c)	10			

Teilaufgabe d)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl (AFB)**	**EK**	**ZK**	**DK**
1	vervollständigt den Quelltext der Methode `ladeWindradtypen`.	8 (II)			
Sachlich richtige Lösungsalternative zur Modelllösung: (8) …….. ……..					
	Summe Teilaufgabe d)	8			

Teilaufgabe e)

	Anforderungen	**Lösungsqualität**			
	Der Prüfling	**maximal erreichbare Punktzahl (AFB)**	**EK**	**ZK**	**DK**
1	analysiert die Methode `liefereEtwas`, indem diese auf die gegebene Belegung der Liste angewendet wird und schrittweise den Aufbau der Liste `auswahlTypen` dokumentiert.	3 (II)			
2	erläutert schrittweise den Aufbau der Liste `auswahlTypen`.	3 (II)			
3	erläutert die Aufgabe der Methode `liefereEtwas` im Sachzusammenhang.	3 (II)			
4	beurteilt, ob das gegebene SQL-Statement des zweiten Ansatzes dieselben Datensätze ermittelt, die die Methode zurückgibt.	4 (III)			
Sachlich richtige Lösungsalternative zur Modelllösung: (13) …….. ……..					
	Summe Teilaufgabe e)	13			