Probeklausur 06.03.2026

Aufgabe 1 (47.25 Punkte)

Implementiere die Klassen

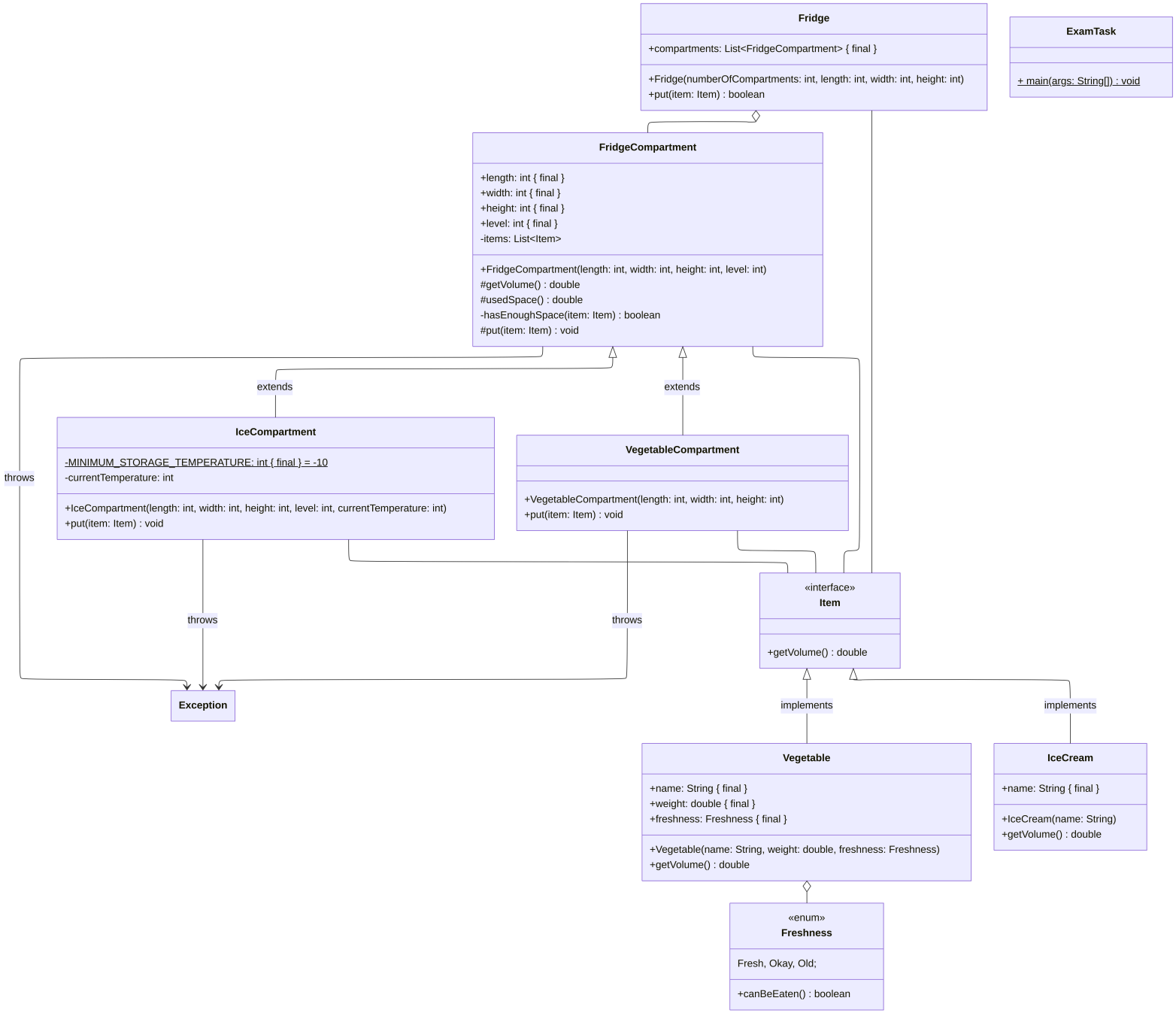
- **Item** (1.5 Punkte),
- **Freshness** (3.5 Punkte),
- **Vegetable** (3.5 Punkte),
- **FridgeCompartment** (10.0 Punkte),
- **IceCompartment** (5.75 Punkte),
- **VegetableCompartment** (6.0 Punkte),
- **Fridge** (12.0 Punkte) und
- **ExamTask** (5.0 Punkte)

entsprechend dem Klassendiagramm.

Befolge alle Hinweise bei der Implementierung!

Glossar

Englisch	**Deutsch**
compartment	Fach
freshness	Frische
fridge	Kühlschrank
ice compartment	Gefrierfach
ice cream	Eiscreme
space	Raum
vegetable	Gemüse
vegetable compartment	Gemüsefach
volume	Volumen


Fridge
+compartments: List<FridgeCompartment> { final }
+Fridge(numberOfCompartments: int, length: int, width: int, height: int)
+put(item: Item) : boolean
ExamTask
+ main(args: String[]) : void
FridgeCompartment
+length: int { final }
+width: int { final }
+height: int { final }
+level: int { final }
-items: List<Item>
+FridgeCompartment(length: int, width: int, height: int, level: int)
#getVolume() : double
#usedSpace() : double
-hasEnoughSpace(item: Item) : boolean
#put(item: Item) : void
extends
extends
throws
IceCompartment
-MINIMUM_STORAGE_TEMPERATURE: int { final } = -10
-currentTemperature: int
+IceCompartment(length: int, width: int, height: int, level: int, currentTemperature: int)
+put(item: Item) : void
VegetableCompartment
+VegetableCompartment(length: int, width: int, height: int)
+put(item: Item) : void
«interface»
Item
+getVolume() : double
throws
throws
Exception
implements
implements
Vegetable
+name: String { final }
+weight: double { final }
+freshness: Freshness { final }
+Vegetable(name: String, weight: double, freshness: Freshness)
+getVolume() : double
IceCream
+name: String { final }
+IceCream(name: String)
+getVolume() : double
«enum»
Freshness
Fresh, Okay, Old;
+canBeEaten() : boolean

Hinweise zur Klasse Freshness

- Die Methode **canBeEaten** soll true zurückgeben, wenn abhängig von der Frische etwas gegessen werden kann.

  Etwas kann gegessen werden, sofern es frisch oder okay ist.

Hinweise zur Klasse Vegetable

- Der **Konstruktor** soll alle Attribute initialisieren.
- Die Methode **getVolume** soll das Volumen zurückgeben.

  Das Volumen eines Gemüses ist die Hälfte des Gewichts.

Hinweise zur Klasse FridgeCompartment

- Der **Konstruktor** soll alle Attribute initialisieren.
- Die Methode **getVolume** soll das Volumen eines Faches berechnen.

  Das Volumen eines Faches ist abhängig von der Länge, Breite und Höhe.

- Die Methode **usedSpace** soll den bereits benutzten Platz eines Faches berechnen.

  Der benutzte Platz ist das Volumen aller Items eines Faches.

- Die Methode **hasEnoughSpace** soll ermitteln, ob das eingehende Item in das Fach passt.

  Ein Item hat genügend Platz in einem Fach, sobald das Volumen des eingehenden Items kleiner ist als der restlich vorhandene Platz eines Faches.

- Die Methode **put** soll versuchen ein Item im Fach unterzubringen.

  Hat das zu platzierende Item genug Platz, soll es dem Fach hinzugefügt werden. Andernfalls soll ein Fehler ausgelöst werden, dass kein Platz mehr vorhanden ist.

Hinweise zur Klasse IceCompartment

- Der **Konstruktor** soll alle Attribute initialisieren.

  Das Eisfach gefriert an den Innenseiten um 5 Einheiten zu und hat somit andere Maße.

- Die Methode **put** soll versuchen, ein Item im Fach unterzubringen.

  Nur Eiscreme darf in das Fach untergebracht werden. Ist das Gefrierfach wärmer als die Mindesttemperatur, darf nichts im Fach untergebracht werden.

  Kann das Item nicht untergebracht werden, soll ein Fehler ausgelöst werden, dass nur Eis in einem kalten Fach untergebracht werden darf.

Hinweise zur Klasse VegetableCompartment

- Der **Konstruktor** soll alle Attribute initialisieren.

  Das Gemüsefach soll immer unten sein (Level 1).

- Die Methode **put** soll versuchen, ein Item im Fach unterzubringen.

  Nur essbares Gemüse darf in das Fach untergebracht werden.

  Kann das Item nicht untergebracht werden, soll ein Fehler ausgelöst werden, dass nur essbares Gemüse untergebracht werden darf.

Hinweise zur Klasse Fridge

- Der **Konstruktor** soll alle Attribute initialisieren.

  Jeder Kühlschrank hat mehrere Fächer. Es gibt jedoch immer genau ein Gemüsefach und ein Gefrierfach. Der Konstruktor soll nach der Initialisierung aller Attribute zuerst das Gefrierfach, dann das Gemüsefach und anschließend N normale Fächer erzeugen und der internen Liste hinzufügen. N entspricht numberOfCompartments. Das Gemüsefach hat immer das geringste Level (beginnend bei 1) und das Gefrierfach immer das höchste Level. Das Gefrierfach hat eine Temperatur von -5.

  Beispiel Liste mit N = 5

  ```
  - Gefrierfach   - Level = 7
  - Gemüsefach    - Level = 1
  - normales Fach - Level = 6
  - normales Fach - Level = 5
  - normales Fach - Level = 4
  - normales Fach - Level = 3
  - normales Fach - Level = 2
  ```

- Die Methode **put** soll ein Item platzieren und zurückgeben, ob es funktioniert hat.

  Versuche einen Platz für das Item im Kühlschrank zu finden. Wenn ein Platz gefunden wurde, soll auf der Konsole die Ebene ausgegeben werden.

  Wenn ein Item in einem Fach nicht platziert werden konnte, soll der Grund auf der Konsole ausgegeben werden.

Hinweise zur Klasse ExamTask

Es soll ein Kühlschrank mit 4 normalen Fächern erstellt werden, welche jeweils 90 lang, 60 breit und 90 hoch sind.

Erstelle ein Eis mit einem Namen deiner Wahl. Erstelle zwei Gemüseobjekte. Das erste hat ein Gewicht von 500, das zweite ein Gewicht von 250. Das leichtere ist alter Rosenkohl, das schwerere ist eine frische Gurke.

Versuche alles im Kühlschrank unterzubringen. Falls kein Fach gefunden wurde, soll dies auf der Konsole ausgegeben werden.

Java API

Klasse	Methode	Statisch	Rückgabetyp
`Freshness`	`values()`	X	`Freshness[]`
`Freshness`	`name()`		`String`
`Exception`	`getMessage()`		`String`
`Boolean`	`valueOf(s: String)`	X	`Boolean`
`Boolean`	`valueOf(b: boolean)`	X	`Boolean`
`Double`	`valueOf(s: String)`	X	`Double`
`Double`	`valueOf(d: double)`	X	`Double`
`Integer`	`valueOf(s: String)`	X	`Integer`
`Integer`	`valueOf(i: int)`	X	`Integer`
`String`	`charAt(index: int)`		`char`
`String`	`length()`		`int`

Java Collections Framework

Klasse	Methode	Statisch	Rückgabetyp
`ArrayList<T>`	`add(element: T)`		`boolean`
`ArrayList<T>`	`add(index: int, element: T)`		`void`
`ArrayList<T>`	`contains(element: T)`		`boolean`
`ArrayList<T>`	`get(index: int)`		`T`
`ArrayList<T>`	`remove(index: int)`		`T`
`ArrayList<T>`	`remove(element: T)`		`boolean`
`ArrayList<T>`	`size()`		`int`
`Collections`	`sort(list: List<T>)`	X	`void`
`Collections`	`sort(list: List<T>, c: Comparator<T>)`	X	`void`

Schnittstellen

Klasse	Methode	Statisch	Rückgabetyp
`Comparable<T>`	`compareTo(o: T)`		`int`
`Comparator<T>`	`compare(o1: T, o2: T)`		`int`