Probeklausur 27.03.2026

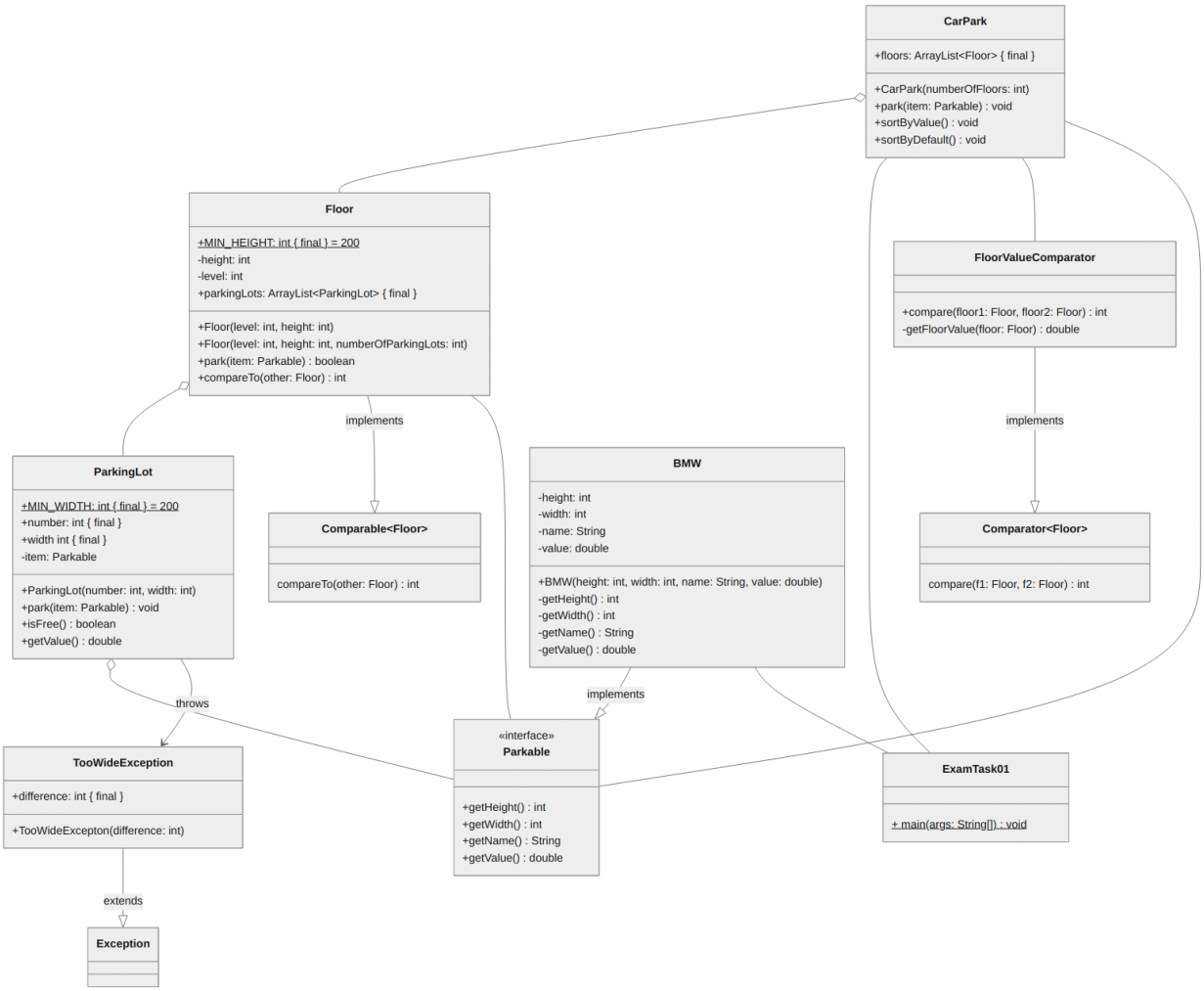
Aufgabe 1 (45 Punkte)

Implementiere die Klassen **Parkable** (2.5 Punkte), **ParkingLot** (8.5 Punkte), **TooWideException** (2 Punkte), **Floor** (13 Punkte), **FloorValueComparator** (6 Punkte), **CarPark** (7 Punkte) und **ExamTask01** (6 Punkte) entsprechend dem Klassendiagramm. Befolge alle Hinweise bei der Implementierung!

Glossar

Englisch	Deutsch
CarPark	Parkhaus
difference	Unterschied
Floor	Etage
height	Höhe
isFree	ist frei
level	Ebene
MIN_HEIGHT	minimale Höhe
MIN_WIDTH	minimale Breite
Parkable	parkbar
Parking Lot	Parkplatz
TooWideException	Zu-Breit-Ausnahme
width	Breite

Klassendiagramm



Hinweise zur Klasse ParkingLot (Parkplatz)

- Der Konstruktor soll alle Attribute initialisieren.
- Die Methode **park** soll das eingehende Parkable parken. Ist das eingehende Parkable breiter (width) als der Parkplatz (ParkingLot) soll eine TooWideException geworfen werden, welche die Differenz des eingehenden Parkable und des Parkplatzes enthält. Ist das Parkable nicht zu breit, darf es dem Parkplatz zugewiesen werden.
- Die Methode **isFree** soll true zurückgeben, wenn kein Parkable im Parkplatz parkt.
- Die Methode **getValue** soll den Fahrzeugwert des Parkable (value) zurückgeben. Ist kein Parkable im Parkplatz, soll 0 zurückgegeben werden. Ist ein Parkable im Parkplatz geparkt, soll der Wert des Parkable zurückgegeben werden.

Hinweise zur Klasse TooWideException (ZuBreitAusnahme)

- Der Konstruktor soll alle Attribute initialisieren.

Hinweise zur Klasse Floor (Etage)

- Die Konstruktoren sollen alle Attribute initialisieren. Rufe im unspezifischen Konstruktor den spezifischen Konstruktor so auf, dass immer 2 Parkplätze (ParkingLots) erzeugt werden.

  Der spezifische Konstruktor soll nach der Initialisierung aller Attribute N Parkplätze erzeugen. N entspricht numberOfParkingLots. Die Parkplätze sollen beginnend mit 1 aufsteigend nummeriert werden. Der erste Parkplatz soll die kleinstmögliche Breite (MIN_WIDTH) haben. Jeder weitere generierte Parkplatz soll 10 Einheiten breiter sein.

  Alle generierten Parkplätze sollen der Etage hinzugefügt werden.

- Die Methode **park** soll das eingehende Parkable parken.

  Ist das eingehende Parkable höher oder gleich hoch (height) wie die Etage, soll false zurückgegeben werden.

  Andernfalls soll das Parkable im ersten freien Parkplatz geparkt werden. Falls es geparkt werden konnte, soll der Name des Parkable, die aktuelle Ebene (level) und die Parkplatznummer (ParkingLot number) in der Konsole ausgegeben werden. Gib anschließend true zurück.

  Bsp: “BMW 320i geparkt. Ebene: 2 Platz: 3”

  Falls das Auto nicht geparkt werden konnte, soll die Differenz, die aktuelle Ebene (level) und die Parkplatznummer (ParkingLot number) in der Konsole ausgegeben werden. Konnte das Auto in keinem Parkplatz geparkt werden, soll false zurückgegeben werden.

  Bsp: “Parkplatz um 20 zu klein. Ebene: 2 Platz: 3”

- Die Methode **compareTo** soll die natürliche Ordnung der Klasse Floor definieren. Hierbei soll nach der Ebene (level) aufsteigend sortiert werden.

Hinweise zur Klasse FloorValueComparator

- Der FloorValueComparator soll das Comparator Interface implementieren und Etagen absteigend nach Etagenwert sortieren.
- Die Methode **getFloorValue** soll den Etagenwert einer Etage berechnen. Dieser setzt sich aus der Summe aller Parkplatzwerte (ParkingLot value) zusammen.

Hinweise zur Klasse CarPark (Parkhaus)

- Der Konstruktor soll alle Attribute initialisieren. Der Konstruktor soll nach der Initialisierung aller Attribute N Etagen (Floors) erzeugen. N entspricht numberOfFloors. Die Etagen sollen beginnend mit 0 als Ebene aufsteigend nummeriert werden. Die erste Etage soll die kleinstmögliche Höhe (MIN_HEIGHT) haben. Jede weitere generierte Etage soll 25 Einheiten höher sein. Alle generierten Etagen sollen dem Parkhaus hinzugefügt werden.
- Die Methode **park** soll das eingehende Parkable in einer Etage (Floor) parken. Konnte das Parkable in einer Etage geparkt werden, soll die Suche in weiteren Etagen abgebrochen werden.
- Die Methode **sortByValue** soll die Etagen nach Etagenwert absteigend sortieren.
- Die Methode **sortByDefault** soll die Etagen nach ihrer natürlichen Ordnung sortieren.

Hinweise zur Klasse ExamTask01

Es soll ein Parkhaus (CarPark) mit fünf Etagen erstellt werden. Anschließend sollen folgende BMWs darin geparkt werden:

Höhe	Breite	Name	Wert
100	100	“BMW 320i”	30.000
250	200	“BMW 340i”	60.000

Sortiere die Etagen nach Etagenwert und finde das wertvollste Parkable der wertvollsten Etage. Gib den Wert des wertvollsten Parkable in der Konsole aus.

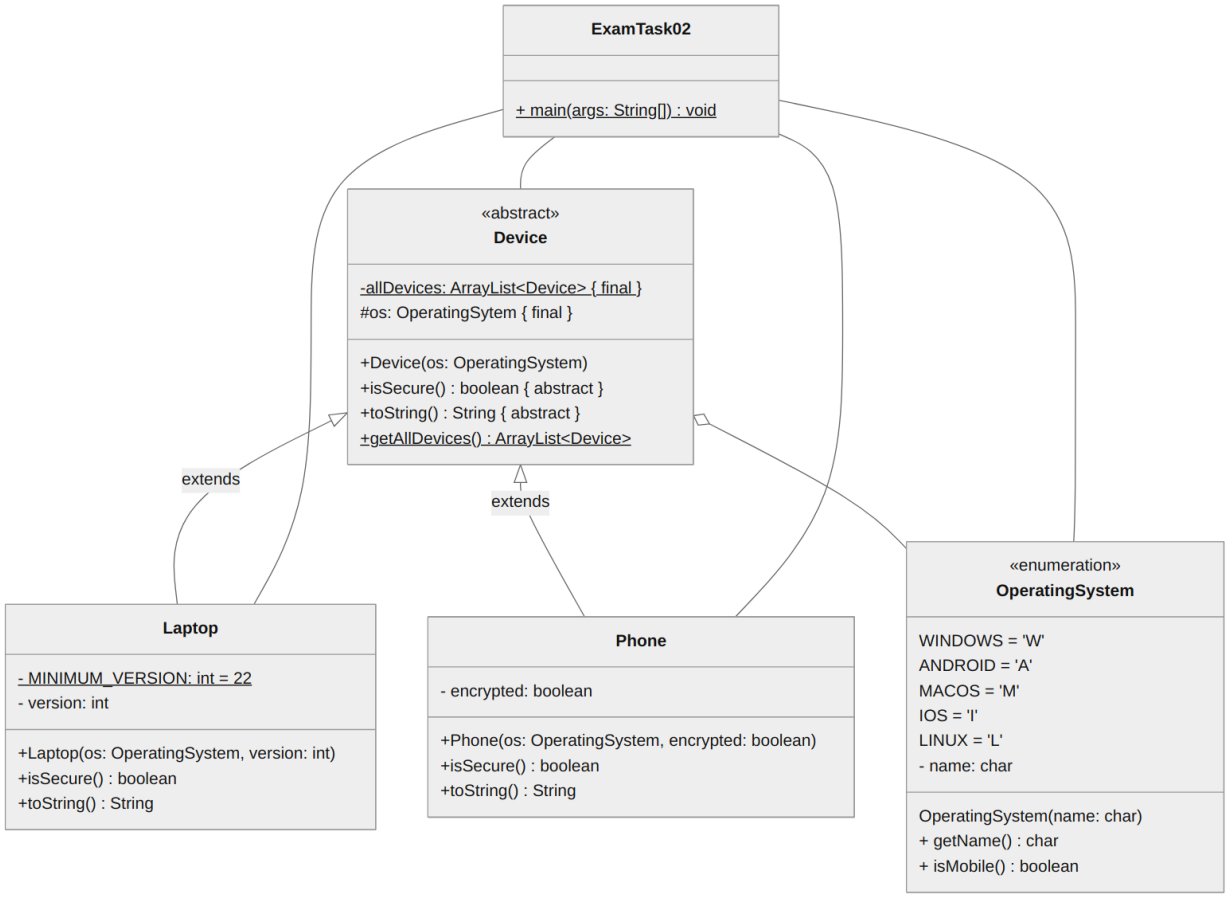
Aufgabe 2 (30 Punkte)

Erstelle die Klassen **OperatingSystem** (5 Punkte), **Device** (5 Punkte), **Phone** (6.5 Punkte), **Laptop** (6.5 Punkte) und **ExamTask02** (7 Punkte) anhand des abgebildeten Klassendiagramms. Befolge alle Hinweise bei der Implementierung!

Glossar

Englisch	Deutsch
Device	Gerät
Operating System	Betriebssystem
Secure	sicher
Phone	Handy
encrypted	verschlüsselt

Klassendiagramm



Hinweise zur Klasse OperatingSystem (Betriebssystem)

- Erstelle die fünf Konstanten Windows, Android, MacOS, iOS und Linux für die Betriebssysteme.
- Der Konstruktor soll alle Attribute initialisieren.
- Die Methode **getName** soll den Namen des Betriebssystems zurückgeben.
- Die Methode **isMobile** soll true zurückgeben, wenn eine Konstante ein Android oder iOS Betriebssystem ist.

Hinweise zur Klasse Device (Gerät)

- Der Konstruktor soll os initialisieren und das erstellte Gerät der ArrayList allDevices hinzufügen.
- Die Methode **getAllDevices** soll die Liste der erstellten Geräte zurückgeben.

Hinweise zur Klasse Phone (Handy)

- Der Konstruktor soll alle Attribute initialisieren.
- Die Methode **isSecure** soll true zurückgeben, wenn ein Handy sicher ist. Ein Handy gilt als sicher, wenn es encrypted ist. Ein Handy gilt auch als sicher, wenn es ein iOS Betriebssystem hat.
- Die Methode **toString** soll alle Attribute und ob es sicher ist in Form eines Strings zurückgeben.

  Bsp: "Phone [encrypted=true] [isSecure=true] [os=I]"

Hinweise zur Klasse Laptop

- Der Konstruktor soll alle Attribute initialisieren.
- Die Methode **isSecure** soll true zurückgeben, wenn der Laptop sicher ist. Ein Laptop gilt als sicher, wenn er kein Windows Betriebssystem hat. Hat ein Laptop Windows als Betriebssystem, gilt er als sicher, sobald die Version dieses Laptops größer als die MINIMUM_VERSION ist.
- Die Methode **toString** soll alle Attribute und ob er sicher ist in Form eines Strings zurückgeben.

  Bsp: "Laptop [version=23] [isSecure=true] [os=W]"

Hinweise zur Klasse ExamTask02

Erzeuge ein Android und ein iOS Handy. Beide sollen nicht verschlüsselt (encrypted) sein. Erzeuge zwei Laptops mit der Version 11. Ein Laptop hat Linux als Betriebssystem, der andere hat Windows als Betriebssystem.

Ermittle jeweils die Anzahl der sicheren Handys und Laptops mithilfe einer Schleife. Gib die jeweilige Anzahl in der Konsole aus. Bsp.: "Laptops: 4 Phones: 3"

Java API

Klasse	Methode	Statisch	Rückgabetyp
`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`
`String`	`split(regex: String)`		`String[]`
`String`	`toLowerCase()`		`String`
`String`	`toUpperCase()`		`String`

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`