Probeklausur 22.05.2026

Aufgabe 1 (65,25 Punkte)

Implementiere folgende Klassen

- **Profile** (7.5 Punkte),
- **Person** (3.0 Punkte),
- **Area** (3.0 Punkte),
- **Flower** (14.5 Punkte),
- **Garden** (27,75 Punkte),
- **ExamTask** (9.5 Punkte),

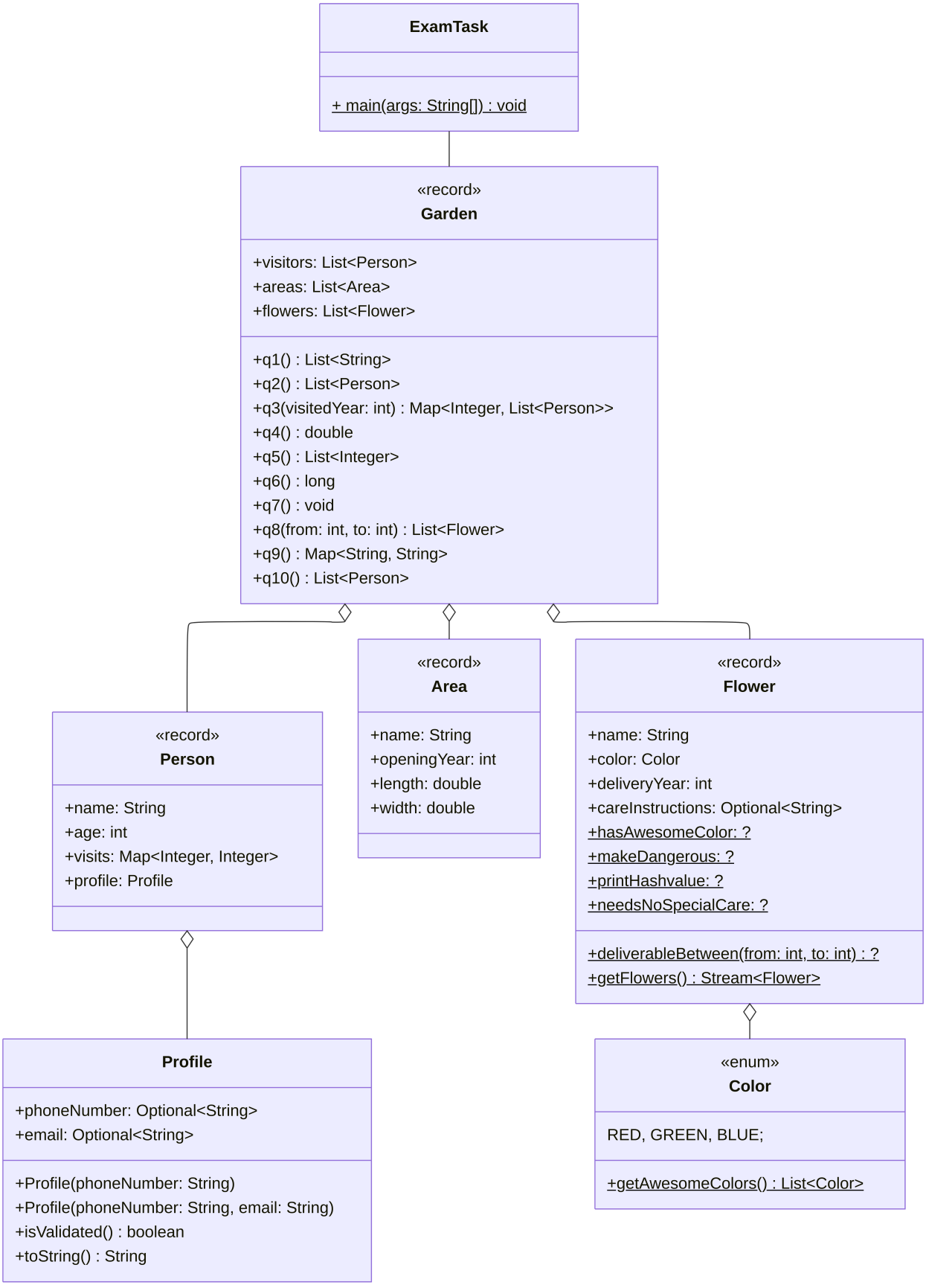
entsprechend dem Klassendiagramm.

Befolge alle Hinweise bei der Implementierung!

Glossar

Englisch	**Deutsch**
Area	Bereich
Awesome Color	Tolle Farbe
Care Instructions	Pflegeanleitung
Delivery Year	Lieferjahr
Flower	Blume
Garden	Garten
Opening Year	Eröffnungsjahr
Profile	Profil
Validated	Validiert
Visitor	Besucher
Visits	Besuche

Klassendiagramm



Hinweise zur Klasse Color

Die Enum-Klasse Color definiert die Farben RED, GREEN und BLUE. Zusätzlich existiert eine statische Methode **getAwesomeColors**, welche eine Liste der tollen Farben zurückgibt.

Hinweise zur Klasse Profile

- Die **Konstruktoren** sollen alle Attribute initialisieren.

  Verwende im unspezifischen Konstruktor den spezifischen Konstruktor wieder. Null Referenzen sollen für alle Parameter des spezifischen Konstruktors erlaubt sein.

- Die Methode **isValidated** soll ermitteln, ob eines der beiden Attribute einen Wert enthält.
- Die Methode **toString** soll das Objekt als String zurückgeben.

  Es sollen nur die Attribute berücksichtigt werden, welche einen Wert haben. Haben beide Attribute einen Wert, sollen die Werte durch ein Leerzeichen getrennt werden. Es sollen ausschließlich die **Werte** berücksichtigt werden.

  Beispiel

  ```
  +49123456

  +49789876 a@b.de

  z@y.de
  ```

Hinweise zur Klasse Area

Jeder Bereich kann durch einen Namen, ein Eröffnungsjahr, eine Länge und Breite beschrieben werden.

Hinweise zur Klasse Flower

Im Klassendiagramm sind nicht alle Datentypen angegeben. Verwende für alle fehlenden Datentypen ein passendes funktionales Interface.

- Das Attribut **hasAwesomeColor** soll eine Lambdafunktion enthalten, die ermittelt, ob die Farbe einer Blume eine tolle Farbe ist. Die Color Klasse definiert alle tollen Farben.
- Das Attribut **makeDangerous** soll eine Lambdafunktion enthalten, die eine bestehende Blume in eine gefährliche Blume umwandelt. Bei der Umwandlung soll die Farbe auf Rot geändert und das Jahr auf 2025 festgelegt werden. Die Pflegeanleitung soll beibehalten werden.
- Das Attribut **printHashvalue** soll eine Lambdafunktion enthalten, welche den String "Hash:" und den Hash Code einer Blume durch ein Leerzeichen getrennt auf der Konsole ausgibt.

  Beispiel

  ```
  Hash: 123456789
  ```

- Das Attribut **needsNoSpecialCare** soll eine Lambdafunktion enthalten, die ermittelt, ob eine Blume keine Pflegeanleitung benötigt.
- Die Methode **deliverableBetween** soll eine Lambdafunktion zurückgeben, die abhängig von den Parametern überprüft, ob eine Blume innerhalb des Start-(from) und Enddatums(to) lieferbar ist.
- Die Methode **getFlowers** soll einen Stream mit einer Blume mit frei gewählten Werten zurückgeben.

Hinweise zur Klasse Person

Jeder Besucher kann durch einen Namen und ein Alter beschrieben werden. Zudem werden die Besuche jedes Besuchers erfasst. Der Schlüssel der Map entspricht dem Jahr des Besuchs und der Wert der Map der Anzahl an Besuchen. Zusätzlich hat jeder Besucher ein Profil.

Hinweise zur Klasse Garden

Der Garten enthält Besucher (visitors), Bereiche (areas) und Blumen (flowers). Benutze die Java Stream API bei der Implementierung der Methoden.

- **q1** Der Inhaber des Gartens möchte seinen ältesten Kundenstamm wertschätzen und sie persönlich ausrufen.

  ```
  Die Methode soll die Namen der zehn ältesten Besucher des Gartens, aufsteigend
  nach ihrem Alter sortiert, in Großbuchstaben als Liste zurückgeben.
  ```

- **q2** Der Inhaber des Gartens ist in Geschenkelaune und möchte seinen größten Fans eine Reise zur Blumeninsel schenken.

  ```
  Die Methode soll die Besucher ermitteln, welche in mindestens 2 Jahren
  jeweils mehr als 3 Besuche hatten.
  ```

- **q3** Der Inhaber des Gartens ist verrückt und stellt eine sinnfreie Anforderung.

  ```
  Die Methode soll alle Besucher ermitteln, welche abhängig vom Parameter
  visitedYear den Garten besucht haben und diese nach Alter gruppieren.
  ```

- **q4** Der Garten hat sein 10 jähriges Jubiläum! Alle Bereiche im Garten, die 10 Jahre oder älter sind, sollen eine Renovierung erhalten.

  ```
  Die Methode soll die Gesamtfläche aller Bereiche im Garten ermitteln, welche
  vor 2015 eröffnet wurden. Die Fläche eines Bereichs ist die Länge mal die Breite.
  ```

- **q5** Der Inhaber des Gartens will wissen, welches Alter in seinem Garten vertreten ist, damit er entscheiden kann, ob er einen PowerBank- oder Rollator-Verleih eröffnen soll.

```
Die Methode soll das jeweilige Alter der Besucher ermitteln und dabei keine
doppelten Werte enthalten.
```

- **q6** Der Inhaber des Gartens leidet an Größenwahn wegen des Jubiläums. Er will wissen, wieviele **Besuche** der Garten seit Eröffnung hatte.

```
Die Methode soll alle Besuche aller Besucher aller Jahre aufsummieren.
```

- **q7** Der Gärtner hat Angst vor giftigen Pflanzen und will gefährliche Blumen finden.

```
Die Methode soll alle Blumen mit tollen Farben filtern, diese in gefährliche
Blumen umwandeln und den Hashwert jeder Blume auf der Konsole ausgeben.
```

- **q8** Der Gärtner will wissen, welche Blumen in einem bestimmten Zeitraum lieferbar sind und keine besondere Pflege benötigen.

```
Die Methode soll alle Blumen ermitteln, welche abhängig von den Parametern from und
to
innerhalb des Lieferjahres liegen und keine Pflegeanleitung haben.
```

- **q9** Der Gärtner will die Pflegeanleitungen aller Blumen haben.

```
Die Methode soll alle Blumen mit ihrem Namen als Schlüssel und ihrer
Pflegeanleitung als Wert zurückgeben. Hat eine Blume keine Pflegeanleitung,
soll ein leerer String verwendet werden.
```

- **q10** Der Inhaber will nur validierte Besucher ansprechen.

```
Die Methode soll alle Besucher ermitteln, welche ein validiertes Profil haben.
```

Hinweise zur Klasse ExamTask

Erstelle die Besucherin Alice mit dem Alter 45. Den Garten hat Sie im Jahr 2023 5 mal besucht und im Jahr 2024 2 Mal besucht. In ihrem Profil ist die Telefonnummer `11880` und die Mail `a@b.de` hinterlegt.

Erstelle den Besucher Bob mit dem Alter 31. Den Garten hat er im Jahr 1999 4 mal besucht und im Jahr 2024 5 mal besucht. In seinem Profil ist nur die Telefonnummer "0815" hinterlegt.

Erstelle folgende Bereiche:

- Rose Garden (Eröffnungsjahr: 2010, Länge: 10.0, Breite: 15.0)
- Cactus Section (Eröffnungsjahr: 2012, Länge: 5.0, Breite: 5.0)

Erstelle eine Liste von Blumen.

Erstelle einen Garten mit den zuvor erstellten Besuchern, Bereichen und Blumen.

Gib abschließend alle Blumen mit ihren Pflegehinweisen auf der Konsole aus.

Aufgabe 2 (5 Punkte)

Wie unterscheidet sich die LinkedList von der ArrayList. Welche Datenstrukturen werden für die jeweiligen Implementierungen verwendet? Mit welchem Datentyp kann ich ohne Anpassungen die Implementierung austauschen?

Aufgabe 3 (5 Punkte)

Was ist die Landau-Notation (Big-O)? Nenne zwei mögliche Komplexitäten und erläutere diese. Welche zwei Ressourcen-Komplexitäten von Algorithmen beschreibt man typischerweise mit der Landau-Notation?

Java API

Methoden der Klasse Optional<T>

Modifier	Methode	Rückgabetyp
static	`of(value: T)`	`Optional<T>`
static	`ofNullable(value: T)`	`Optional<T>`
static	`empty()`	`Optional<T>`
instance	`isPresent()`	`boolean`
instance	`isEmpty()`	`boolean`
instance	`ifPresent(Consumer<T> consumer)`	`void`
instance	`ifPresentOrElse(Consumer<T> present, Consumer<T> empty)`	`void`
instance	`get()`	`T`
instance	`orElse(T other)`	`T`

Funktionale Interfaces

Interface	Methode	Rückgabetyp
`Predicate<T>`	`test(T value)`	`boolean`
`Function<T, R>`	`apply(T value)`	`R`
`Consumer<T>`	`accept(T value)`	`void`
`Supplier<T>`	`get()`	`T`
`Comparator<T>`	`compare(T item1, T item2)`	`int`

Methoden der Klasse Stream<T>

Modifier	Methode	Rückgabetyp
static	`of(T… values)`	`Stream<T>`
instance	`allMatch(Predicate<T> predicate)`	`boolean`
instance	`anyMatch(Predicate<T> predicate)`	`boolean`
instance	`noneMatch(Predicate<T> predicate)`	`boolean`
instance	`count()`	`long`
instance	`collect(Collector<T,A,R> collector)`	`R`

Modifier	Methode	Rückgabetyp
instance	`distinct()`	`Stream<T>`
instance	`filter(Predicate<T> predicate)`	`Stream<T>`
instance	`findAny()`	`Optional<T>`
instance	`findFirst()`	`Optional<T>`
instance	`forEach(Consumer<T> consumer)`	`void`
instance	`limit(long maxSize)`	`Stream<T>`
instance	`map(Function<T, R> mapper)`	`Stream<R>`
instance	`mapToDouble(Function<T, Double> mapper)`	`DoubleStream`
instance	`mapToInt(Function<T, Integer> mapper)`	`IntStream`
instance	`mapToLong(Function<T, Long> mapper)`	`LongStream`
instance	`max(Comparator<T> comparator)`	`Optional<T>`
instance	`min(Comparator<T> comparator)`	`Optional<T>`
instance	`skip(long n)`	`Stream<T>`
instance	`sorted(Comparator<T> comparator)`	`Stream<T>`
instance	`toList()`	`List<T>`

Methoden der Statistik Klassen (IntStream, LongStream, DoubleStream)

Modifier	Methode	Rückgabetyp
instance	`sum()`	`long`
instance	`average()`	`OptionalDouble`

Methoden der Klasse Collectors

Modifier	Methode	Rückgabetyp
static	`groupingBy(Function<T, U> mapper)`	`Map<U, List<T>>`
static	`toMap(Function<T, U> keyMapper, Function<T, V> valueMapper)`	`Map<U, V>`

Methoden des Interfaces Collection<T>

Modifier	Methode	Rückgabetyp
instance	`stream()`	`Stream<T>`

Methoden der Klasse Integer

Modifier	Methode	Rückgabetyp
static	`valueOf(String value)`	`int`
static	`compare(Integer i1, Integer i2)`	`int`

Methoden der Klasse Double

Modifier	Methode	Rückgabetyp
static	`valueOf(String value)`	`Double`
static	`compare(Double d1, Double d2)`	`int`

Methoden der Klasse Long

Modifier	Methode	Rückgabetyp
static	`valueOf(String value)`	`Long`
static	`compare(Long l1, Long l2)`	`int`

Methoden des Interfaces Map<K,V>

Modifier	Methode	Rückgabetyp
instance	`put(key: K, value: V)`	`V`
instance	`keySet()`	`Set<K>`
instance	`values()`	`Collection<V>`
instance	`entrySet()`	`Set<Map.Entry<K, V>>`

Methoden der Schnittstelle List<E>

Modifier	Methode	Rückgabetyp
static	`of(E… items)`	`List<E>`

Methoden der Klasse Object

Modifier	Methode	Rückgabetyp
instance	`hashCode()`	`int`