

# Probeklausur 27.02.2026

# Aufgabe 1 (51,5 Punkte)

Implementiere die Klassen

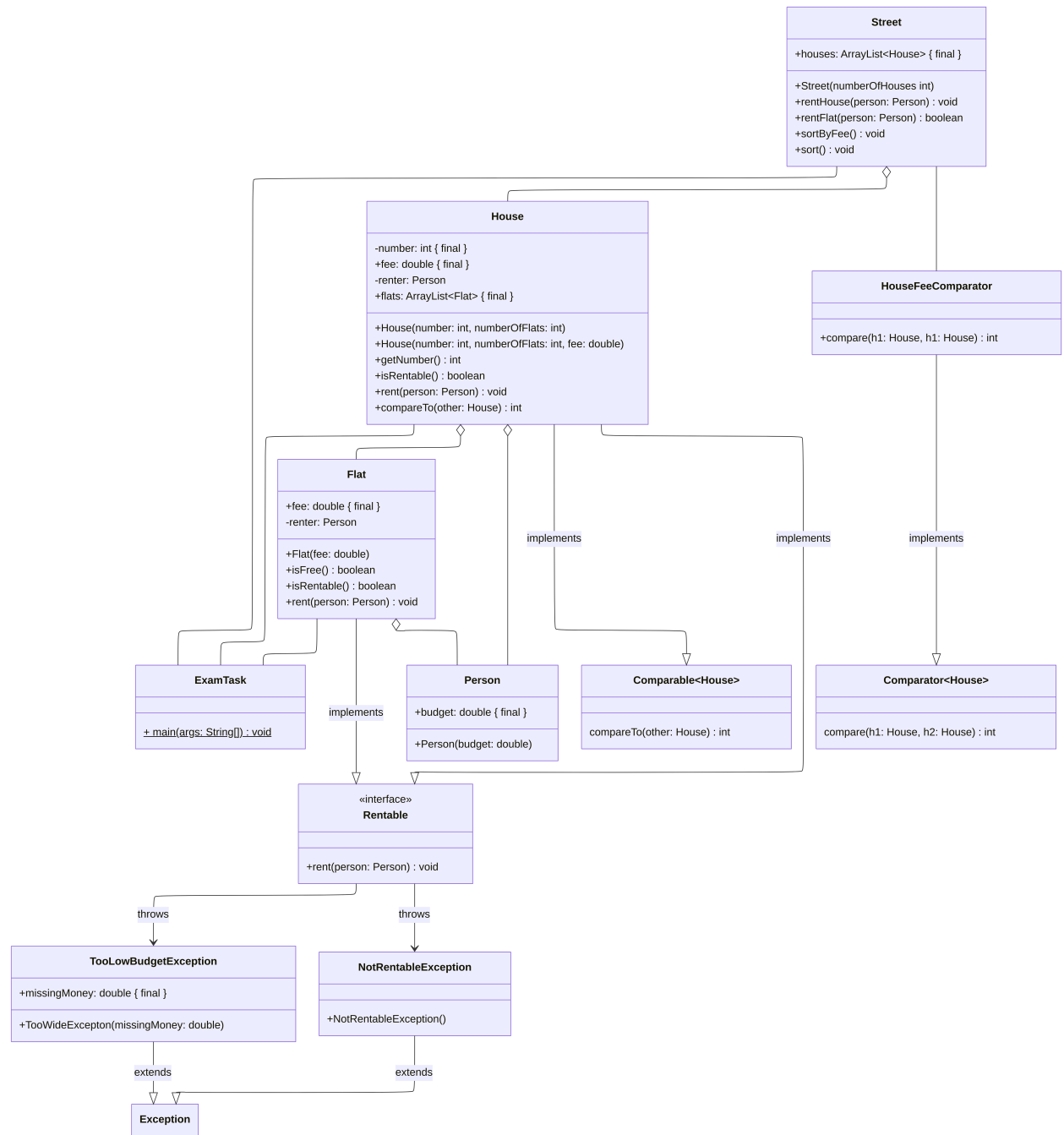
- **Rentable** (2 Punkte),
- **NotRentableException** (1.5 Punkte),
- **TooLowBudgetException** (2 Punkte),
- **Flat** (7.5 Punkte),
- **House** (14.5 Punkte),
- **HouseFeeComparator** (2.75 Punkte),
- **Street** (16 Punkte) und
- **ExamTask** (5.25 Punkte)

entsprechend dem Klassendiagramm.

Befolge alle Hinweise bei der Implementierung!

## Glossar

| Englisch     | Deutsch       |
|--------------|---------------|
| House        | Haus          |
| Number       | Nummer        |
| Renter       | Mieter        |
| Street       | Straße        |
| TooLowBudget | Zu wenig Geld |



## Hinweise zur Klasse Rentable

- Die Methode **rent** soll angeben, dass diese Methode eine **NotRentableException** oder eine **TooLowBudgetException** auslösen kann.

## Hinweise zur Klasse TooLowBudgetException

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

## Hinweise zur Klasse Flat

- Der **Konstruktor** soll alle Attribute initialisieren.
- Die Methode **isFree** soll true zurückgeben, wenn kein Mieter in der Wohnung ist.
- Die Methode **isRentable** soll true zurückgeben, wenn die Gebühr größer als 0 ist.
- Die Methode **rent** soll die eingehende Person mieten lassen.

Ist die Wohnung nicht mietbar oder nicht frei, soll eine **NotRentableException** ausgelöst werden.

Ist die Wohnung mietbar und frei, aber nicht bezahlbar für die eingehende Person soll eine **TooLowBudgetException** ausgelöst werden. Eine Wohnung ist nicht bezahlbar, sobald die Gebühr größer ist als das Budget einer Person.

Ist die Wohnung mietbar, frei und bezahlbar soll die eingehende Person als Mieter zugewiesen werden.

## Hinweise zur Klasse House

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

Rufe im unspezifischen Konstruktor den spezifischen Konstruktor so auf, dass immer 0 Gebühren für das Haus gesetzt werden.

Der spezifische Konstruktor soll nach der Initialisierung aller Attribute N Wohnungen erzeugen. N entspricht numberOfFlats. Die erste Wohnung soll eine Gebühr von 500 haben. Jede weitere generierte Wohnung soll 100 Einheiten teurer sein.

Alle generierten Wohnungen sollen dem Haus hinzugefügt werden.

- Die Methode **getNumber** soll die Hausnummer zurückgeben.
- Die Methode **isRentable** soll true zurückgeben, wenn die Gebühr größer als 0 ist und kein Mieter im Haus ist.
- Die Methode **rent** soll die eingehende Person mieten lassen.

Ist das Haus nicht mietbar, soll eine **NotRentableException** ausgelöst werden.

Ist das Haus mietbar, aber nicht bezahlbar für die eingehende Person soll eine **TooLowBudgetException** ausgelöst werden. Ein Haus ist nicht bezahlbar, sobald die Gebühr

---

größer ist als das Budget einer Person.

Ist das Haus mietbar und bezahlbar soll die eingehende Person als Mieter zugewiesen werden.

- Die Methode **compareTo** soll die natürliche Ordnung der Klasse House definieren. Hierbei soll nach der Hausnummer aufsteigend sortiert werden.

## Hinweise zur Klasse HouseFeeComparator

- Der HouseFeeComparator soll das Comparator Interface implementieren und Häuser aufsteigend nach Gebühr sortieren.

## Hinweise zur Klasse Street

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

Der Konstruktor soll nach der Initialisierung aller Attribute N Häuser erzeugen. N entspricht numberOfHouses. Die Häuser sollen beginnend mit 1 aufsteigend nummeriert werden. + Häuser mit einer geraden Hausnummer, sollen 5 Wohnungen haben jedoch keine Gebühr. Häuser mit einer ungeraden Hausnummer, sollen keine Wohnungen haben, jedoch eine Gebühr von 2000.

Verwende für die Instanziierung von Häusern, wenn möglich einen unspezifischen Konstruktor.

Füge die generierten Wohnungen der Straße hinzu.

- Die Methode **rentHouse** soll die eingehende Person in einem Haus mieten lassen. + Konnte die Person ein Haus mieten, soll die Suche in weiteren Häusern abgebrochen werden.

Kann ein Haus nicht gemietet werden, weil das Budget nicht ausreicht, soll dies auf der Konsole ausgegeben werden. Relevant ist das fehlende Geld.

*Beispiel*

"Es wird 22.3 mehr Geld gebraucht."

Kann ein Haus aus anderen Gründen nicht gemietet werden, soll dies auf der Konsole ausgegeben werden. Relevant ist Hausnummer.

*Beispiel*

"Die Hausnummer 11 kann nicht gemietet werden."

- Die Methode **rentFlat** soll die eingehende Person in irgendeiner Wohnung in irgendeinem Haus mieten lassen.

Konnte die Person eine Wohnung mieten, soll true zurückgegeben werden.

---

Kann eine Wohnung nicht gemietet werden, weil das Budget nicht ausreicht, soll dies auf der Konsole ausgegeben werden. Relevant ist die Gebühr.

*Beispiel*

"Zu wenig Geld für Wohnung. Gebühr: 500.0"

Kann eine Wohnung aus anderen Gründen nicht gemietet werden, soll dies auf der Konsole ausgegeben werden. Relevant ist die wievielte Wohnung im Haus, beginnend bei 1.

*Beispiel*

"Wohung 1 nicht Mietbar"

Kann keine Wohnung in keinem Haus der Straße von der eingehenden Person gemietet werden, soll false zurückgegeben werden.

- Die Methode **sortByFee** soll die Häuser nach Gebühr aufsteigend sortieren.
- Die Methode **sort** soll die Häuser nach ihrer natürlichen Ordnung sortieren.

## Hinweise zur Klasse ExamTask

Es soll eine Straße mit 20 Häusern erstellt werden.

Sortiere die Häuser aufsteigend nach ihrer Gebühr und finde die günstigste Wohnung der gesamten Straße.

Gib anschließend die Gebühr der günstigsten Wohnung aus.

---

## Aufgabe 2 (5 Punkte)

Was ist der Unterschied zwischen if-else und switch? Erläutern Sie die Unterschiede und Restriktionen. Nennen Sie einen konkreten Anwendungsfall und begründen Sie ihre Entscheidung.

# Java API

| Klasse    | Methode              | Statisch | Rückgabetyp |
|-----------|----------------------|----------|-------------|
| Enum      | values()             | X        | Enum[]      |
| Enum      | 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         |
| String    | split(regex: String) |          | String[]    |
| String    | toUpperCase()        |          | String      |
| String    | toLowerCase()        |          | 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         |