Probeklausur

Einführung in die Informatik

Universität:	Technische Universität Berlin
Kurs/Modul:	Einführung in die Informatik
Bearbeitungszeit:	120 Minuten
Erstellungsdatum:	September 19, 2025

Bearbeitungszeit: 120 Minuten.

Aufgabe 1.

(a) Beschreiben Sie ein einfaches Klassenmodell für eine Bibliothek mit den Bereichen Buch, Exemplar und Benutzer. Geben Sie mindestens die Attribute an, die jeweils sinnvoll wären, und beschreiben Sie, welche Methoden Sie pro Klasse vorsehen würden.

(b) Erklären Sie den Unterschied zwischen Klassen, Objekten und Instanzen. Geben Sie ein praktisches Beispiel aus dem Bereich Informatikunterricht.

(c) Was versteht man unter Vererbung in der Objektorientierung? Skizzieren Sie ein Beispiel mit einer Oberklasse `Person` und zwei Unterklassen `Student` und `Dozent`. Beschreiben Sie, welche Eigenschaften vererbt werden und was eine Überschreibung von Methoden bedeutet.

(d) Beschreiben Sie die Konzepte der Polymorphie und der Kapselung. Nennen Sie jeweils ein konkretes Beispiel aus einer Programmiersprache Ihrer Wahl.

Aufgabe 2.

(a) Erklären Sie den Unterschied zwischen primitiven Datentypen und Referenztypen. Geben Sie Beispiele für beide Konzepte in einer typischen OO-Sprache.

(b) Diskutieren Sie das Konzept von Arrays als Sammlungen von Elementen ersten Typs. Welche Eigenschaften besitzen Arrays hinsichtlich Länge, Zugriff und Speicherlayout.

(c) Was versteht man unter statischen Methoden versus Instanzmethoden? Wann ist die Nutzung einer statischen Methode sinnvoll?

(d) Beschreiben Sie den Unterschied zwischen Schleifen und Rekursion. Nennen Sie zwei Vor- und zwei Nachteile beider Ansätze.

Aufgabe 3.

(a) Beschreiben Sie den typischen Ablauf einer Informationssuche in einem Recherchesystem. Welche Schritte gehören dazu (z. B. Eingabe der Suchanfrage, Indexierung, Ranking, Darstellung der Ergebnisse)?

(b) Skizzieren Sie ein kleines Objektmodell zur Repräsentation von Dokumenten in einer Suchanwendung. Welche Attribute wären sinnvoll, welche Beziehungen existieren?

(c) Welche Suchoperatoren kennen Sie und wie beeinflussen sie die Ergebnisse einer Abfrage?

(d) Welche Parameter beeinflussen das Ranking von Suchergebnissen, und warum ist die Berücksichtigung von Relevanz wichtig?

Aufgabe 4.

(a) Erklären Sie den Unterschied zwischen Prozessen und Threads in einem Betriebssystem. Welche Vorteile ergeben sich durch Multithreading?

(b) Beschreiben Sie zwei gängige Scheduling-Verfahren: First-Come-First-Served und Round-Robin. Welche Vor- und Nachteile haben sie in Bezug auf Reaktionszeit und Durchsatz?

(c) Was versteht man unter dem Unterschied zwischen Hauptspeicher und Cache? Welche Rolle spielen Speicherhierarchie und Speicherzugriffe für die Systemleistung?

(d) Nennen Sie drei zentrale Aufgaben eines Betriebssystems und erläutern Sie kurz, wie diese umgesetzt werden.

Lösungen

Bearbeitungszeit: 120 Minuten.

Aufgabe 1.

(a) Beschreibung des Klassenmodells (Bibliothek)

Es wird ein einfaches, nachvollziehbares objektorientiertes Modell vorgeschlagen, das die drei Kernelemente Buch, Exemplar und Benutzer abbildet und über eine optionale Ausleihe-Beziehung (Ausleihe) verknüpft. Die Entscheidungen zielen darauf ab, Trennung von Buchdaten (Metadaten) und Leihobjekten zu ermöglichen.

- **Klasse Buch**
  - Attribute: isbn (String), titel (String), autor (String), verlag (String), jahr (int), seiten (int), sprache (String), kategorie (String)
  - Methoden:
    - `getTitel()`, `getAutor()`, `getISBN()`, `gibVerfuegbareExemplare()` – liefert eine Liste der Exemplar-Objekte dieses Buches, die aktuell verfügbar sind
    - `sucheExemplare()` – Suche nach passenden Exemplaren in der Bibliothek (basierend auf Status, Standort, etc.)
- **Klasse Exemplar**
  - Attribute: id (String bzw. long), buch (Buch), zustand (Enum Zustand: VERFUEGBAR, AUSGELIEHEN, DEFECT, VERLOREN), standort (String)
  - Methoden:
    - `istVerfuegbar()` – true, falls zustand == VERFUEGBAR
    - `ausleihen(Benutzer)` – markiert Exemplar als AUSGELIEHEN und erzeugt ggf. eine Ausleihe
    - `Rueckgabe()` – setzt Zustand zurück auf VERFUEGBAR (bzw. status aktualisieren)
    - `getBuch()` – Referenz auf das zugehörige Buch
- **Klasse Benutzer**
  - Attribute: benutzerId (String), name (String), vorname (String), email (String), telefon (String), adresse (String)
  - Methoden:
    - `ausleihen(Exemplar e)`, `rueckgabe(Exemplar e)`
    - `getAusleihen()` – Liste der aktuell ausgeliehenen Exemplare
- **Optionale Hilfsklasse: Ausleihe**
  - Attribute: ausleihId (String), benutzer (Benutzer), exemplar (Exemplar), datumAusleihe (Date), datumRueckgabe (Date), status (Enum: OFFEN, ABGESCHLOSSEN, VERLAENGERT)
  - Methoden: `verlaengern()`, `istAbgeschlossen()`

Die Beziehungen: `Buch` besitzt eine oder mehrere `Exemplar`-Instanzen; `Ausleihe` verbindet `Benutzer` und `Exemplar` und modelliert den Leihvorgang. Diese Trennung erleichtert z. B. die Suche nach verfügbaren Exemplaren unabhängig von den Buchtaten.

(b) Unterschied zwischen Klassen, Objekten und Instanzen

- **Klasse** (Class): eine Blaupause oder ein Bauplan, der Attribute und Methoden definiert. Beispiel: `Klasse Buch` mit Feldern isbn, titel, autor.
- **Objekt** (Object) bzw. `Instanz` einer Klasse: eine konkrete Ausprägung der Blaupause mit Werten, z. B. ein Buch mit isbn="978-3-16-148410-0", titel="Musterbuch".
- **Instanz** ist synonym zu Objekt in diesem Kontext und bezeichnet eine konkrete Repräsentation einer Klasse, d. h. eine konkrete `Buch`-Ausprägung.

Beispiel aus dem Informatikunterricht: Die Klasse `Kurs` bildet die Blaupause für Kurse (Feldwerte wie Kursname, Kursnummer). Eine konkrete Instanz dieses Bauplans könnte `Kurs: INFO101, Titel "Einführung in Informatik"` sein.

(c) Vererbung in der Objektorientierung

- **Oberklasse** `Person` mit gemeinsamen Attributen wie `name`, `vorname`, `geburtsdatum`, `adresse`.
- **Unterklassen** `Student` und `Dozent` erben diese Attribute und ergänzen spezialisierte Eigenschaften.
  - **Student**: `matrikelnr`, `studiengang`
  - **Dozent**: `lehrgebiet`, `dienststelle`
- **Vererbung** bedeutet, dass Unterklassen die Felder und Methoden der Oberklasse übernehmen und erweitern können.
- **Überschreibung (Override)**: Eine Methode in der Oberklasse kann in der Unterklasse neu implementiert werden, z. B. `beschreibung()` in `Person`, die in `Student` anders ausfällt (zusätzliche Studieninformationen) bzw. in `Dozent` (zusätzliche Lehrdaten).

Beispiel (Java-ähnlich):

```
class Person {
  String name;
  String vorname;
  String beschreibung() { return name + ", " + vorname; }
}
class Student extends Person {
  String matrikelnummer;
  String studiengang;
  @Override
  String beschreibung() { return super.beschreibung() +
      " Student: " + matrikelnummer + " " + studiengang; }
}
class Dozent extends Person {
  String lehrgebiet;
```

```
  @Override
  String beschreibung() { return super.beschreibung() +
      " Dozent: " + lehrgebiet; }
}
```

(d) Polymorphie und Kapselung

- **Polymorphie**: Ein Objekt kann durch Referenzen einer Oberklasse behandelt werden, während die konkrete Implementierung der Unterklasse zur Laufzeit bestimmt wird.
  - Beispiel: Eine Liste von `Person`-Referenzen enthält `Student`- und `Dozent`-Objekte. Aufruf von `beschreibung()` erzeugt je nach konkrete Klasse den passenden Satz.
- **Kapselung (Encapsulation)**: Implementierungsdetails werden versteckt; Felder werden (typischerweise) privat gehalten; öffentliche Getter/Setter steuern den Zugriff.
  - Beispiel (Java-ähnlich): `class Konto` mit `private double kontostand;` `public void einzahlen(double amount)` und `public void auszahlen(double amount)` zum kontrollierten Zugriff.

```
List<Person> personen = new ArrayList<>();
personen.add(new Student(...));
personen.add(new Dozent(...));
for (Person p : personen) {
  System.out.println(p.beschreibung()); // polymorpher Aufruf
}
```

Aufgabe 2.

(a) Primitive Datentypen vs Referenztypen

- **Primitive Typen** (in vielen OO-Sprachen wie Java): `int`, `long`, `double`, `boolean` etc. Sie speichern Werte direkt.
- **Referenztypen** (Objekte): `String`, `Array`, eigene Klasseninstanzen. Sie speichern Zeiger auf Objekte.
- **Beispiele (Java-ähnlich)**:
  - primitive: `int n = 5;`
  - referenziert: `String s = "Hallo";` bzw. `String r = new String("Hallo");`

(b) Arrays als Sammlungen von Elementen ersten Typs

- Arrays sind typisiert (homogen): alle Elemente haben denselben Typ.
- Eigenschaften: feste Länge `length`, direkter Indexzugriff mit `array[i]` (0-basiert), konstante Zugriffzeit O(1).
- Speicherlayout: meist zusammenhängender Speicher (speicherliche Kontiguität) mit direktem Zeigerzugriff auf Elemente.

- In vielen OO-Sprachen sind Arrays selbst Objekte (Referenztypen).

(c) Statische Methoden vs. Instanzmethoden

- **Statische Methoden** (class methods): gehören zur Klasse, lassen sich ohne Objekt aufrufen, z. B. `Math.max(a,b)`.
- **Instanzmethoden** (object methods): gehören zu einer konkreten Instanz eines Objekts, z. B. `str.toLowerCase()`.
- **Wann sinnvoll?**
  - Statisch: Hilfsfunktionen ohne Objektzustand, Fabrikmethoden, Factory-Pattern, Utility-Funktionen.
  - Instanzmethoden: arbeiten auf dem Zustand eines Objekts, z. B. Methoden, die Felder lesen/verändern.

(d) Unterschied zwischen Schleifen und Rekursion

- **Schleifen**
  - Vorteile: geringerer Overhead, transparente Speicher-/Stack-Nutzung, in vielen Fällen schneller.
  - Nachteile: kann bei komplexen Problemen unübersichtlich werden.
- **Rekursion**
  - Vorteile: elegante, klare Lösungen (z. B. Baum- bzw. Teilproblemlösungen).
  - Nachteile: Stack-Raumbedarf, Risiko des Stack-Overflows bei zu tiefer Rekursion, oft langsamer durch Funktionsaufrufe.

Aufgabe 3.

(a) Typischer Ablauf einer Informationssuche in einem Recherchesystem

- Eingabe der Suchanfrage durch den Nutzer
- Vorverarbeitung der Abfrage (Normalisierung, Tokenisierung, Stopwort-Entfernung, ggf. Stemming)
- Indizierung/Indexzugriff: Abruf der relevanten Dokumente aus dem Inverted Index
- Ranking/Bewertung: Anwendung von Relevanzkennzahlen (z. B. TF-IDF, BM25)
- Darstellung der Ergebnisse: Listendarstellung, Snippets, Relevanz-Werte
- Interaktives Feedback und ggf. Refinement der Suche (erweiterte Suche)

(b) Kleines Objektmodell zur Repräsentation von Dokumenten in einer Suchanwendung

- **Dokument** (Dokument)
  - Attribute: `id`, `titel`, `autor`, `inhalt`, `datum`, `dateiFormat`

  - Beziehungen: besitzt `Metadaten`, kann `Indexeinträge` referenzieren

- **Metadaten** (Metadata)
  - Attribute: `sprache`, `länge`, `format`, `quelle`
- **Indexeintrag** (IndexEntry)
  - Attribute: `term`, `termFrequency`, `documentId`, `positionen`

(c) Suchoperatoren und deren Einfluss auf das Abfrageergebnis

- **AND** (UND): Dokumente müssen alle angegebenen Begriffe enthalten
- **OR** (ODER): Dokumente müssen mindestens einen der Begriffe enthalten
- **NOT** (NICHT): Dokumente, die einen bestimmten Begriff enthalten, werden ausgeschlossen
- **Phrasen- bzw. Nähe-Anfragen**: exakte Reihenfolge oder räumliche Nähe der Begriffe (z. B. "Künstliche Intelligenz" nahe beieinander)
- **Klammern** setzen: Priorität der Operatoren kontrollieren
- **Wildcard** (`*`, `?`): erweiterte Term-Suche (z. B. `Kle*` für Klett, Klebe, Kleinen)

(d) Parameter, die das Ranking von Suchergebnissen beeinflussen, und Relevanz

- **Term Frequency** (TF): Häufigkeit eines Begriffs im Dokument
- **Inverse Document Frequency** (IDF): Seltenheit eines Begriffs im Korpus
- **Dokumentlänge** (Normalization): längere Dokumente erhalten oft eine Anpassung, damit sie nicht bevorzugt werden
- **Feldergewichtung**: z. B. Begriffe im Titel stärker gewichten
- **Aktualität/Recency**: jüngere Dokumente werden ggf. höher bewertet
- **Klick-/Nutzungsaktivität** (Relevanz-Signale): Benutzerinteraktion kann Ranking beeinflussen

Aufgabe 4.

(a) Prozesse vs. Threads in einem Betriebssystem

- **Prozess**: eigenständiger Ausführungszustand mit eigenem Adressraum; Kontextwechsel ist teurer
- **Thread**: Leichtgewicht-Thread innerhalb eines Prozesses; teilen sich Adressraum und Ressourcen des Prozesses
- **Multithreading-Vorteile**: erhöhte Parallelität/Throughput, bessere Auslastung von CPU-Kernen, verbesserte Responsiveness bei GUI-Anwendungen

- **Herausforderungen**: Synchronisation, Deadlocks, Race Conditions; Kontextwechsel zwischen Threads ist günstiger als zwischen Prozessen

(b) Zwei Scheduling-Verfahren: FCFS und Round-Robin

- **First-Cit-First-Served (FCFS)**: Prozesse werden in der Reihenfolge ihrer Ankunft bedient
  - Vorteile: einfach, vorhersehbares Verhalten
  - Nachteile: problematisch bei langen Prozessen, kann lange Wartezeiten anderer Prozesse verursachen (Intransparenz)
- **Round-Robin (RR)**: Zeit-Slices (Quants) werden zyklisch zugewiesen
  - Vorteile: faire Verteilung der CPU-Zeit, gute Reaktionszeit für interaktive Prozesse
  - Nachteile: kontextwechselintensiv, bei sehr kleinen Timeslices Overhead

(c) Unterschied Hauptspeicher vs. Cache

- **Hauptspeicher** (RAM): großer, langsamerer Speicher, global zugänglich
- **Cache**: kleine, extrem schnelle Speicherebene nahe der CPU
- **Speicherhierarchie** reduziert Latenz durch Lokalisierungsprinzipien (lokale Temporalität, räumliche Lokalität)
- **Speicherzugriffe**: Cache-Kohärenz muss bei mehreren Kernen/Prozessen beachtet werden

(d) Drei zentrale Aufgaben eines Betriebssystems und kurze Erläuterung

- **Prozess- und Thread-Verwaltung**: Erzeugung/Ausführung, Scheduling, Synchronisation
- **Speicherverwaltung**: Zuweisung von Hauptspeicher, Paging/Segmentierung, Speicherschutz
- **I/O-Management (Geräte- und Dateisystem-Management)**: Treiber, Pufferung, Dateisystemlogik, Zugriffskontrollen