**Kapitel 1: (60 Fragen)**

1. Was versteht man unter einem Programmierparadigma?

2. Wozu dient ein Berechnungsmodell?

3. Welche Berechnungsmodelle werden in Programmierparadigmen verwendet, und welche charakteristischen Eigenschaften haben sie?

4. Welche Eigenschaften von Berechnungsmodellen sind für deren Erfolg häufig (mit)bestimmend?

5. Im Spannungsfeld welcher widersprüchlichen Ziele befinden sich Programmierparadigmen? Wie äußert sich dieses Spannungsfeld?

6. Was ist die strukturierte Programmierung? Wozu dient sie?

7. Wie gehen unterschiedliche Paradigmen mit Seiteneffekten um?

8. Was bedeutet referentielle Transparenz, und wo findet man referentielle Transparenz?

9. Wieso passt referentielle Transparenz nicht gut mit Ein- und Ausgabe zusammen, und wie kann man das Dilemma lösen?

10. Welchen Zusammenhang gibt es zwischen Seiteneffekten und der objektorientierten Programmierung?

11. Was sind First-Class-Entities? Welche Gründe sprechen für deren Verwendung, welche dagegen?

12. Was haben Funktionen höherer Ordnung mit einem applikativen Programmierstil zu tun?

13. Welche Modularisierungseinheiten gibt es, was sind ihre charakteristischen Eigenschaften, und wodurch unterscheiden sie sich?

14. Welche Bedeutung haben Schnittstellen für Modularisierungseinheiten? Warum unterscheidet man zwischen von außen zugreifbaren und privaten Inhalten?

15. Was ist und wozu dient ein Namensraum?

16. Warum können Module nicht zyklisch voneinander abhängen, Komponenten aber schon?

17. Was versteht man unter Datenabstraktion, Kapselung und DataHiding?

18. Warum und inwiefern ist die Einbindung von Komponenten komplizierter als die von Modulen?

19. Wie kann man globale Namen verwalten?

20. Was versteht man unter Parametrisierung? Wann kann das Befüllen von „Lücken“ durch welche Techniken erfolgen?

21. Warum braucht man zur Parametrisierung in der Objekterzeugung neben Konstruktoren gelegentlich auch Initialisierungsmethoden?

22. Welche Vor- und Nachteile hat die zentrale Ablage von Werten zum Zweck der Parametrisierung?

23. Was unterscheidet Generizität von den verschiedenen Formen der Parametrisierung zur Laufzeit?

24. Was sind Annotationen und wozu kann man sie verwenden? Wodurch unterscheiden sie sich von Generizität?

25. Was versteht man unter aspektorientierter Programmierung?

26. Wodurch unterscheidet sich Parametrisierung von der Ersetzbarkeit, und warum ist die Ersetzbarkeit von so zentraler Bedeutung?

27. Wann ist A durch B ersetzbar?

28. Wodurch kann festgelegt sein, ob A durch B ersetzbar ist?

29. Was ist die Signatur einer Modularisierungseinheit?

30. Wie verhält sich die Signatur einer Modularisierungseinheit zur Abstraktion, die durch diese Modularisierungseinheit gebildet wird?

31. Was sind Zusicherungen, und welche Rolle spielen sie für Modularisierungseinheiten?

32. Wann sind Typen miteinander konsistent, und was sind Typfehler?

33. Wie schränken Typen die Flexibilität ein, und warum verwendet man Typen trotzdem?

34. Welche Gründe sprechen für den Einsatz statischer Typprüfungen, welche dagegen?

35. Was versteht man unter Typinferenz? Welche Gründe sprechen für bzw. gegen den Einsatz?

36. Zu welchen Zeitpunkten können Entscheidungen getroffen werden (Typen und Entscheidungsprozesse)?

37. Welchen Einfluss können Typen auf Entscheidungszeitpunkte haben?

38. Wie beeinflussen Typen die Planbarkeit weiterer Schritte?

39. Was ist ein abstrakter Datentyp?

40. Was unterscheidet strukturelle von nominalen Typen?

41. Warum verwenden wir in Programmiersprachen meist nominale Typen, in theoretischen Modellen aber hauptsächlich

strukturelle?

42. Wie hängen Untertypbeziehungen mit Ersetzbarkeit zusammen?

43. Warum kann ein Compiler ohne Unterstützung durch Programmierer(innen) nicht entscheiden, ob ein nominaler Typ

Untertyp eines anderen nominalen Typs ist?

44. Erklären Sie Einschränkungen bei Untertypbeziehungen zusammen mit statischer Typprüfung.

45. In welchem Zusammenhang verwendet man Higher-Order-Subtyping und F-gebundene Generizität?

46. Wie konstruiert man rekursive Datenstrukturen?

47. Was versteht man unter Fundiertheit rekursiver Datenstrukturen? Welche Ansätze dazu kann man unterscheiden?

48. Warum wird Typinferenz in objektorientierten Sprachen meist nur lokal beschränkt eingesetzt?

49. Wie können statisch geprüfte Typen beliebige Eigenschaften von Werten propagieren?

50. Erklären Sie folgende Begriffe:

• Objekt, Klasse, Vererbung

• Identität, Zustand, Verhalten, Schnittstelle

• deklarierter, statischer und dynamischer Typ

• Faktorisierung, Refaktorisierung

• Verantwortlichkeiten, Klassenzusammenhalt, Objektkopplung

51. Welche Arten von Polymorphismus unterscheidet man? Welche davon sind in der objektorientierten Programmierung wichtig? Warum?

52. Wann sind zwei gleiche Objekte identisch und wann sind zwei identische Objekte gleich?

53. Sind Datenabstraktion, Datenkapselung und Data-Hiding einander entsprechende Begriffe? Wenn Nein, worin unterscheiden sie sich?

54. Was besagt das Ersetzbarkeitsprinzip? (Häufige Prüfungsfrage!)

55. Warum ist Ersetzbarkeit in der objektorientierten Programmierung so wichtig (mehrere Gründe)?

56. Wann und warum ist gute Wartbarkeit wichtig?

57. Wie lauten die wichtigsten Faustregeln im Zusammenhang mit Klassenzusammenhalt und Objektkopplung? Welche Vorteile kann man sich davon erwarten, dass diese Faustregeln erfüllt sind?

58. Welche Arten von Software kann man wiederverwenden, und welche

Rolle spielt jede davon in der Softwareentwicklung?

59. Welche Rolle spielen Refaktorisierungen in der Wiederverwendung?

60. Wofür ist die objektorientierte Programmierung gut geeignet, und

wofür ist sie nicht gut geeignet?

**Kapitel 2: (22 Fragen)**

1. In welchen Formen (mindestens zwei) kann man durch das Ersetzbarkeitsprinzip Wiederverwendung erzielen?

2. Wann ist ein struktureller Typ Untertyp eines anderen strukturellenTyps? Welche Regeln müssen dabei erfüllt sein? Welche zusätzliche Bedingungen gelten für nominale Typen bzw. in Java? (Hinweis: Häufige Prüfungsfrage!)

3. Sind die in Punkt 2 angeschnittenen Bedingungen (sowie das, was Compiler prüfen können) hinreichend, damit das Ersetzbarkeitsprinzip erfüllt ist? Wenn nicht, was muss noch beachtet werden?

4. Was bedeutet Ko-, Kontra- und Invarianz, und für welche Typen in einer Klasse trifft welcher dieser Begriffe zu? (Hinweis: Häufige Prüfungsfrage!)

5. Was sind binäre Methoden, und welche Schwierigkeiten verursachen sie hinsichtlich der Ersetzbarkeit?

6. Wie soll man Typen formaler Parameter wählen um gute Wartbarkeit zu erzielen?

7. Warum ist dynamisches Binden gegenüber switch- oder geschachtelten if-Anweisungen zu bevorzugen?

8. Dient dynamisches Binden der Ersetzbarkeit und Wartbarkeit?

9. Welche Arten von Zusicherungen werden unterschieden, und wer ist für die Einhaltung verantwortlich? (Hinweis: Häufige Prüfungsfrage!)

10. Wie müssen sich Zusicherungen in Unter- und Obertypen zueinander verhalten, damit das Ersetzbarkeitsprinzip erfüllt ist? Warum? (Hinweis: Häufige Prüfungsfrage!)

11. Warum sollen Signaturen und Typen stabil bleiben? Wo ist Stabilität besonders wichtig?

12. Was ist im Zusammenhang mit allgemein zugänglichen (= möglicherweise nicht nur innerhalb des Objekts geschriebenen) Variablen und Invarianten zu beachten?

13. Wie genau sollen Zusicherungen spezifiziert sein?

14. Wozu dienen abstrakte Klassen und abstrakte Methoden? Wo und wie soll man abstrakte Klassen einsetzen?

15. Ist Vererbung das gleiche wie das Ersetzbarkeitsprinzip? Wenn Nein,

wo liegen die Unterschiede?

16. Worauf kommt es zur Erzielung von Codewiederverwendung eher an, auf Vererbung oder Ersetzbarkeit? Warum?

17. Was bedeuten folgende Begriffe in Java?

• Objektvariable, Klassenvariable, statische Methode

• Static-Initializer

• geschachtelte und innere Klasse

• final Klasse und final Methode

• Paket, Class-Path, import-Anweisung

18. Wo gibt es in Java Mehrfachvererbung, wo Einfachvererbung?

19. Welche Arten von import-Deklarationen kann man in Java unterscheiden? Wozu dienen sie?

20. Wozu benötigt man eine package-Anweisung?

21. Welche Möglichkeiten zur Spezifikation der Sichtbarkeit gibt es in Java, und wann soll man welche Möglichkeit wählen?

22. Wodurch unterscheiden sich Interfaces in Java von abstrakten Klassen? Wann soll man Interfaces verwenden? Wann sind abstrakte Klassen besser geeignet?

**Kapitel 3: (19 Fragen)**

1. Was ist Generizität? Wozu verwendet man Generizität?

2. Was ist gebundene Generizität? Was kann man mit Schranken auf Typparametern machen, das ohne Schranken nicht geht?

3. In welchen Fällen soll man Generizität einsetzen, in welchen nicht?

4. Was bedeutet statische Typsicherheit in Zusammenhang mit Generizität, dynamischen Typabfragen und Typumwandlungen?

5. Was sind (gebundene) Wildcards als Typen in Java? Wozu kann man sie verwenden?

6. Welche Arten von Generizität kann man hinsichtlich ihrer Übersetzung und ihrem Umgang mit Schranken unterscheiden?

Welche Art wird in Java verwendet, und wie flexibel ist diese Lösung?

7. Wie kann man Generizität simulieren? Worauf verzichtet man, wenn man Generizität nur simuliert?

8. Was wird bei der heterogenen bzw. homogenen Übersetzung von Generizität genau gemacht?

9. Was muss der Java-Compiler überprüfen um sicher zu sein, dass durch Generizität keine Laufzeitfehler entstehen?

10. Welche Möglichkeiten für dynamische Typabfragen gibt es in Java, und wie funktionieren sie genau?

11. Was wird bei einer Typumwandlung in Java umgewandelt – der deklarierte, dynamische oder statische Typ? Warum?

12. Welche Gefahren bestehen bei Typumwandlungen?

13. Wie kann man dynamische Typabfragen und Typumwandlungen vermeiden? In welchen Fällen kann das schwierig sein?

14. Welche Arten von Typumwandlungen sind sicher? Warum?

15. Was sind kovariante Probleme und binäre Methoden? Wie kann man mit ihnen umgehen oder sie vermeiden?

16. Wie unterscheidet sich Überschreiben von Überladen, und was sind Multimethoden?

17. Wie kann man Multimethoden simulieren? Welche Probleme können dabei auftreten?

18. Was ist das Visitor-Entwurfsmuster?

19. Wodurch ist Überladen problematisch, und in welchen Fällen ergeben sich kaum Probleme?

**Kapitel 4: (15 Fragen)**

1. Wie werden Ausnahmebehandlungen in Java unterstützt?

2. Wie sind Ausnahmen in Untertypbeziehungen zu berücksichtigen?

3. Wozu kann man Ausnahmen verwenden? Wozu soll man sie verwenden, wozu nicht?

4. Durch welche Sprachkonzepte unterstützt Java die nebenläufige Programmierung? Wozu dienen diese Sprachkonzepte?

5. Wozu brauchen wir Synchronisation? Welche Granularität sollen wir dafür wählen?

6. Zu welchen Problemen kann Synchronisation führen, und was kann man dagegen tun?

7. Wozu dienen Annotationen? Wann setzt man sie sinnvoll ein?

8. Wie lange können Annotationen leben? Wofür ist welche Lebensdauer sinnvoll?

9. Wie kann man eigene Annotationen deklarieren? Welche Gemeinsamkeiten und Unterschiede zu Interfaces bestehen?

10. Wie kann man zur Laufzeit auf Annotationen zugreifen?

11. Was ist aspektorientierte Programmierung? Wann setze ich sie sinnvoll ein?

12. Was bedeutet Separation-of-Concerns?

13. Was sind Core-Concerns, was Cross-Cutting-Concerns?

14. Was sind Join-Points, Pointcuts, Advices und Aspekte, und wozu braucht man sie?

15. An welchen Programmpunkten können sich Join-Points befinden?

**Kapitel 5: (14 Fragen)**

1. Erklären Sie folgende Entwurfsmuster und beschreiben Sie jeweils Anwendungsgebiet, Struktur, Eigenschaften und wichtige Details der Implementierung unter Verwendung vorgegebener Namen:

• Decorator

• Factory-Method

• Iterator

• Prototype

• Proxy

• Singleton

• Template-Method

• Visitor (siehe Abschnitt 3.4.2)

2. Welche Arten von Iteratoren gibt es, und wofür sind sie geeignet?

3. Wie wirkt sich die Verwendung eines Iterators auf die Schnittstelle des entsprechenden Aggregats aus?

4. Inwiefern können geschachtelte Klassen bei der Implementierung von Iteratoren hilfreich sein?

5. Was ist ein robuster Iterator? Wozu braucht man Robustheit?

6. Wird die Anzahl der benötigten Klassen im System bei Verwendung von Factory-Method, Prototype, Decorator und Proxy (gegenüber einem System, das keine Entwurfsmuster verwendet) eher erhöht, vermindert oder bleibt sie unverändert?

7. Wird die Anzahl der benötigten Objekte im System bei Verwendung von Factory-Method, Prototype, Decorator und Proxy (gegenüber einem System, das keine Entwurfsmuster verwendet) eher erhöht, vermindert oder bleibt sie unverändert?

8. Vergleichen Sie Factory-Method mit Prototype. Wann stellt welches Entwurfsmuster die bessere Lösung dar? Warum?

9. Wo liegen die Probleme in der Implementierung eines so einfachen Entwurfsmusters wie Singleton?

10. Welche Unterschiede und Ähnlichkeiten gibt es zwischen Decorator und Proxy?

11. Welche Probleme kann es beim Erzeugen von Kopien im Prototype geben? Was unterscheidet flache Kopien von tiefen?

12. Für welche Arten von Problemen ist Decorator gut geeignet, für welche weniger? (Oberfläche versus Inhalt)

13. Kann man mehrere Decorators bzw. Proxies hintereinander verketten? Wozu kann so etwas gut sein?

14. Was unterscheidet Hooks von abstrakten Methoden?