**Das Klassenkonzept**

1. Entspricht eine Klasse eher einem Rezept oder einem Kuchen?
   1. Einem Rezept.
2. Entspricht ein Objekt eher einem Rezept oder einem Kuchen?
   1. Einem Kuchen.
3. Sind Eigenschaften eines Objektes eher Nomen, Verben oder Adjektive?
   1. Adjektive.
4. Sind Methoden einer Klasse eher Nomen, Verben oder Adjektive?
   1. Verben.
5. Ist der Klassenname eher ein Nomen, ein Verb oder ein Adjektiv?
   1. Nomen
6. Wie viele Objekte kann man mit Hilfe einer Klasse erzeugen?
   1. Logisch unbegrenzt viele, physisch durch Speicherplatz begrenzt.
7. Wie viele Klassen benötigt man, um ein Objekt zu erzeugen?
   1. 1-n.
8. Wieso soll man Eigenschaften mit private deklarieren?
   1. Encapsulation garantieren.
9. Wo kann man die Eigenschaften einer Klasse überall verwenden?
   1. Wenn sie public deklariert wurde bzw in ihrem Scope
10. Warum benötigt man setter?
    1. Um das Anpassen von Variablen zu kontrollieren
11. Warum benötigt man getter?
    1. Um das indirekte Zugreifen auf Variablen zu ermöglichen, ohne sie von außen anpassbar zu machen
12. Was ist eine Instanz und warum heißen Eigenschaften auch Instanzvariablen?
13. Teilen sich mehrere Objekte dieselben Instanzvariablen?
14. Was benötigt man in main, damit man eine Methode verwenden kann, die nicht static ist?
15. Wann ist eine Methode static und wann nicht?

**Konstruktoren und Überladung**

1. Durch welche drei Merkmale unterscheiden sich Konstruktoren von normalen Methoden?
2. Definieren Sie Überladen durch drei Merkmale.
3. Gibt es Überladen auch für normale Methoden?
4. An welchen drei Kriterien kann der Rechner unterscheiden, welche überladene Methode bei einem Aufruf benutzt wird?

**Referenzen Speicherveraltung und static**

1. Wie viele verschiedene Werte kann man in einer static Variable (kein Array) speichern?
2. Hilft eine static Variable dabei, die Anzahl von Mitarbeitern in drei verschiedenen Arbeitsgruppen festzuhalten? Warum?
3. Was haben static Methoden und static Variablen gemeinsam?
4. Wie gibt man im Klassendiagramm bekannt, dass etwas static ist?
5. Wieso erkenne ich an einem Aufruf innerhalb der eigenen Klasse nicht, ob es sich um eine static Methode handelt?
6. Woran kann man außerhalb der Klasse in der die Methode implementiert wurde erkennen, ob es eine static oder eine "Instanzmethode" ist?

**Komposition und Aggregation**

1. Fülle aus:   
   Komposition und Aggregation sind beides " \_\_\_\_ ein" Beziehungen und bedeuten, dass die Klasse des \_\_\_\_\_\_\_\_\_\_ eine Eigenschaft vom Typ der Klasse des \_\_\_\_\_\_\_\_\_\_ enthält.
2. Welche Information wird mit Kardinalitäten angegeben?
3. Wie stellt man Komposition im Klassendiagramm dar?   
   Wie stellt man Aggregation im Klassendiagramm dar?   
   Auf welcher Seite steht dieses Symbol?
4. Was sind die Gemeinsamkeiten von Komposition und Aggregation?
5. Was sind die Unterschiede zwischen Komposition und Aggregation?
6. Ist eine Komposition auch eine Assoziation?  
   Ist jede Aggregation auch eine Assoziation?  
   Ist jede Assoziation auch eine Komposition oder Aggregation?

**Vererbung**

1. Welchen Vorteil bringt die Vererbung?
2. Wie sieht der Vererbungspfeil aus und in welche Richtung zeigt er?
3. Mit welchem Schlüsselwort gibt man die Vererbung an und bei welcher Klasse steht es?
4. Füllen Sie aus:

Kompositionen und Aggregationen sind "\_\_\_\_\_\_\_ ein" Beziehungen.  
Vererbungen sind "\_\_\_\_\_\_\_ ein" Beziehungen .

1. Darf man private Eigenschaften von darüber liegenden Klassen verwenden? Wie?
2. Was ist Redefinition?   
   Worin besteht der Unterschied zur Überladung?
3. Darf man einer Referenz einer abgeleiteten (erbenden) Klasse eine Instanz einer darüber liegenden Klasse zuordnen, z.B. Malocher m = new Mitarbeiter();  
   Warum ist das so sinnvoll geregelt? Vervollständigen Sie den folgenden Lückentext als Antwort:

Weil die **Referenz** der \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Klasse, in diesem Fall die Klasse \_\_\_\_\_\_\_\_\_\_\_\_ , bestimmt, welche \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ werden dürfen. Das dahinter liegende eigentliche Objekt \_\_\_\_\_\_\_\_\_\_\_ lediglich diese \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Wenn das Objekt nicht \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ist, kann es die Anforderung \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

1. Welche Konsequenz hat es, wenn eine darüber liegende Klasse keinen default-Konstruktor hat?
2. Wann benötigt man die super Referenz?
3. Worin liegt der Unterschied zwischen super. und super() ?

**Polymorphie und abstrakte Klassen**

1. Was geht und was geht nicht? Warum?  
   Malocher m = new Mitarbeiter();  
   Mitarbeiter m = new Malocher();  
   Malocher m = new Buerohengst();
2. Warum funktioniert eine Methode nur dann polymorph, wenn sie redefiniert wurde?
3. Was ist der Vorteil von Polymorphie?
4. Welche Methoden stehen für ein polymorphes Objekt zur Verfügung? Wer bestimmt das – die Referenz oder die Instanz?
5. Ist "late binding" bzw. "dynamisches Binden" ein Laufzeitvorteil?
6. Welches Schlüsselwort kennzeichnet Methoden, die keine Implementation haben?
7. Wie bezeichnet man Klassen, von denen keine Instanzen gebildet werden können?
8. Welchen Zweck erfüllen abstrakte Methoden?
9. Warum schreibt man abstrakte Klassen?
10. Warum darf man von abstrakten Klassen keine Instanzen bilden?
11. Wie kann ich im UML-Klassendiagramm erkennen, ob eine Klasse oder eine Methode abstrakt ist?
12. Nennen Sie drei Varianten für polymorphe Objekte.
13. Welche Methoden stehen zur Verfügung, wenn man die Referenz einer abstrakten Klasse für die Polymorphie nutzt?

**Packages**

**Das Klassenkonzept**

**Das Klassenkonzept**