JAVA 07: Vererbung und Schnittstellen in Java

panitz

Zusammenfassung

In diesem Kurs sind Fagen zur Programmiersprache Java in Bezug auf Vererbung und Schnittstellen. Weitere Fragen beschäftigen sich mit Sichtbarkeiten und Paketen. Auch Aspekte der Gleichheit werden behandelt.

Frage: Typzusicherung (Cast)

Was realisiert eine Typzusicherung der folgenden Art?

```
String makeString(Object o){
  return (String)o;
}
```

(Eine Antwortmöglichkeit.)

- Dynamische Typüberprüfung
- Statischen Typcheck
- Datenkonvertierung im Speicher
- Erzeugung eines neuen String-Objektes

Korrekte Antworten

- Dynamische Typüberprüfung

Erläuterung

Eine Typzusicherung auch Downcast bezeichnet realisiert einen dynamischen Typcheck, das heißt eine Überprüfung des Typen zur Laufzeit. Schlägt diese Typüberprüfung zur Laufzeit fehl wird eine entsprechende Ausnahme geworfen.

Frage: Konstruktoraufruf

Gegeben seien folgende Klassen:

```
class Upper{
  Upper(){ System.out.print("U ");}
}
class Lower extends Upper{
  Lower(int i){ System.out.print("L1 ");}
  Lower(){
    this(42);
    System.out.print("L2 ");
  } }
```

Welche Ausgabe auf der Kommandozeile ergibt folgender Ausdruck `new Lower()`

(Eine Antwortmöglichkeit.)

- U L1 L2
- U L1
- U L2
- L2
- L1
- L2 L1 U
- L1 L2 U
- U

Korrekte Antworten

- U L1 L2

Erläuterung

Der erste Aufruf in einem Konstruktor muss immer der Aufruf eines Konstruktoren der Oberklasse mit dem Schlüsselwort `super` sein, es sei denn es ist der Aufruf eines anderen Konstruktors der eigenen Klasse mit dem Schlüsselwort `this`.
Steht dieser Aufruf nicht Quelltext, dann generiert dort der Compiler den Aufruf `super()`.
In disem Beispiel wird mit `new Lower()` zunächst der überladene Konstruktor mit `new Lower(42)` aufgerufen. Dieser ruft als erstes den Konstruktor der Oberklasse `new Upper()` auf. Dieser macht die erste Ausgabe `U`. Anschließend kommt die Ausgabe `L1` und schließlich `L2`.

Frage: Methodenterminologie

Was ist hier zu erkennen?

```
class Test{
  int m(String s){
    return s.length;
  }
  int m(int i){
    return i*i;
  }
}
```

(Eine Antwortmöglichkeit.)

- Überladung einer Methode
- Polymorphie
- Generische Methoden
- Überschreibung einer Methode
- Statische Methoden

Korrekte Antworten

- Überladung einer Methode

Erläuterung

Zwei Methoden mit gleichen Namen und unterschiedlichen Parametern in einer Klasse wird als Überladung (overloading) bezeichnet. Bei einer Methode mit gleichem Namen und gleichen Parametertypen wie eine aus einer Oberklasse geerbten Methoden, spricht man von Überschreiben der geerbten Methode.

Frage: Vererbung

Welche der folgenden Aussagen ist korrekt für alle Klassen. Die Klasse `java.lang.Object` sei dabei ausgenommen?

(Mehrere Antwortmöglichkeiten).

- Jede Klasse hat genau eine Oberklasse.
- Klassen haben keine oder genau eine Oberklasse.
- Jede Klasse implementiert mindestens eine Schnittstelle.
- Eine Klasse hat keine oder beliebig viele Unterklassen.
- Eine Klasse hat maximal eine Unterklasse.

Korrekte Antworten

- Jede Klasse hat genau eine Oberklasse.
- Eine Klasse hat keine oder beliebig viele Unterklassen.

Erläuterung

Nur die Klasse `java.lang.Object` hat keine Oberklasse. Alle anderen Klassen haben genau eine Oberklasse.

Frage: Sichtbarkeiten und Konstruktoren

Welche der folgenden Aussagen ist korrekt für alle Klassen.

(Mehrere Antwortmöglichkeiten).

- Jede Klasse hat mindestens einen Konstruktor.
- Jede Klasse hat mindestens einen Konstruktor mit Sichtbarkeitsattribut `public`.
- Abstrakte Klassen haben keine Konstruktoren.
- Schnittstellen haben keine Konstruktoren.
- Schnittstellen haben nur Konstruktoren mit Sichtbarkeitsattribut `public`.

Korrekte Antworten

- Jede Klasse hat mindestens einen Konstruktor.
- Schnittstellen haben keine Konstruktoren.

Erläuterung

Wenn der Programmierer keinen Konstruktor definiert, wird vom Compiler der default-Konstruktor automatisch hinzu gefügt.

Frage: Überschreiben von Methoden

Betrachten Sie folgende Klassen.

```
class A{
  String f(){return "A";}
}
class B extends A{
  @Override String f(){
    return "B";
  }
  public static void main(String[] args){
    B b = new B();
    A a = b;
    System.out.print(b.f());
    System.out.print(a.f());
  }
}
```

Welche Ausgabe erfolgt auf der Kommandozeile beim Ausführen der Klasse B.

Korrekte Antworten

- BB

Erläuterung

Durch die Späte-Bindung (late binding) wird für ein Objekt immer die überschriebene Methode aufgerufen, die die Klasse von der das Objekt erzeugt wurde hat. Unanbhängig davon, von welchen Typ die Referenz auf das Objekt ist.

Frage: Überschreiben von Methoden

Betrachten Sie folgende Klassen.

```
class A{
  String s =  "A";
}
class B extends A{
  String s =  "B";

  public static void main(String[] args){
    B b = new B();
    A a = b;
    System.out.print(b.s);
    System.out.print(a.s);
  }
}
```

Welche Ausgabe erfolgt auf der Kommandozeile beim Ausführen der Klasse B.

Korrekte Antworten

- BA

Erläuterung

Die Späte-Bindung (late binding) funktioniert in Java nur für Methoden nicht für Felder. Felder können im eigentlichen Sinne nicht wie Methoden überschrieben werden.

Frage: Späte-Bindung (late binding)

Welche der folgenden Aussagen ist korrekt in der Programmiersprache Java?

(Mehrere Antwortmöglichkeiten).

- Es gibt keine Späte-Bindung für Felder.
- Späte-Bindung funktioniert nur für Methoden aus Schnittstellen.
- Späte-Bindung funktioniert auch für statische Methoden.
- Späte-Bindung tritt bei nicht-statischen Methoden, die überschrieben wurden, auf.

Korrekte Antworten

- Es gibt keine Späte-Bindung für Felder.
- Späte-Bindung tritt bei nicht-statischen Methoden, die überschrieben wurden, auf.

Erläuterung

Frage: Schnittstelle

Welche der folgenden Aussagen sind korrekt in der Programmiersprache Java?

(Mehrere Antwortmöglichkeiten).

- Eine Klasse kann beliebig viele Schnittstellen implementieren.
- Jede Klasse implemeniert mindestens eine Schnittstelle.
- Eine Schnittstelle darf nur genau eine andere Schnitstelle erweitern.
- Eine Schnittstelle hat maximal eine Methode.
- Eine abstrakte Klasse muss nicht alle Methoden einer implementierten Schnittstelle implementieren.

Korrekte Antworten

- Eine Klasse kann beliebig viele Schnittstellen implementieren.
- Eine abstrakte Klasse muss nicht alle Methoden einer implementierten Schnittstelle implementieren.

Erläuterung

Schnittstellen haben nur abstrakte Methoden, die die Sichtbarkeit `public` haben. Seit Java 1.8 können Schnittstellen zusätzlich noch Standardmethoden (default) haben. Eine Klasse kann beliebig viele Schnittstellen implementieren. Schnitstellen können beliebig viele Oberschnittstellen erweitern.

Frage: Abstrakte Klassen

Welche der folgenden Aussagen sind korrekt in der Programmiersprache Java?

(Mehrere Antwortmöglichkeiten).

- In einer abstrakten Klasse können abstrakte Methoden aufgerufen werden.
- Eine Klasse kann mehrere abstrakte Klassen erweitern.
- Objekte einer abstrakten Klasse können durch den Aufruf des Konstruktors der Klasse über das Schlüsselwort `new` erzeugt werden.
- Eine abstrakte Klasse hat abstrakte Methoden. Zur Laufzeit kann es zu Laufzeitfehlern kommen, wenn diese Methoden nicht implementiert sind.
- Abstrakte Methoden aus Schnittstellen haben immer die Sichtbarkeit `public`.

Korrekte Antworten

- In einer abstrakten Klasse können abstrakte Methoden aufgerufen werden.
- Abstrakte Methoden aus Schnittstellen haben immer die Sichtbarkeit `public`.

Erläuterung

Abstrakte Klassen sind Klassen und es gelten in Bezug auf Vererbung und Konstruktoren dieselben Bedingungen wie für Klassen. Man kann nur keine Objekte von abstrakten Klassen direkt erzeugen, sondern nur von Unterklassen, die nicht mehr abstrakt sind, sondern alle abstrakten Methoden implementieren.

Frage: Konstanten

Welches Schlüsselwort zeigt in Java an, dass ein Variable nach der Initialisierung keinen neuen Wert zugewiesen bekommen darf?

Korrekte Antworten

- final

Erläuterung

Zusätzlich können Methoden `final` sein. Dies bedeutet, sie dürfen nicht überschrieben werden. Auch Klassen können `final` sein. Dieses bedeutet, es darf keine Unterklasse definiert werden.

Frage: Sichtbarkeiten

Wie viele Sichtbarkeiten kennt die Programmiersprache Java für Felder und Methoden? (Anwort als Ziffer.)

Korrekte Antworten

- 4

Erläuterung

Es gibt die drei Sichtbarkeitsattribute: `public`, `protected` und `private`. Die Standardsichtbarkeit ist Paket.

Frage: Aufzählungen

Welche der folgenden Aussagen sind korrekt in der Programmiersprache Java?

(Mehrere Antwortmöglichkeiten).

- Von Aufzählungsklassen gibt es eine feste Anzahl von Objekten.
- Eine Aufzählungslasse hat keinen Konstruktor.
- Es können Unterklassen einer Aufzählungsklasse definiert werden.
- Aufzählungswerte können zur Fallunterscheidung in einemr switch-case-Anweisung verwendet werden.

Korrekte Antworten

- Von Aufzählungsklassen gibt es eine feste Anzahl von Objekten.
- Aufzählungswerte können zur Fallunterscheidung in einemr switch-case-Anweisung verwendet werden.

Erläuterung

Eine Aufzählungsklasse (enum) hat nur einen privaten Konstruktor. Es gibt gibt nur endlich viele Objekte, die in statischen Variablen gehalten werden. Aufzählungsklassen sind final, so dass keine Unterkjlassen geschrieben werden können. Eine switch-case-Anweisung kann Aufzählungsobjekte zur Fallunterscheidung nutzen.

Frage: Gleichheit

Welche der folgenden Aussagen sind korrekt in der Programmiersprache Java?

(Mehrere Antwortmöglichkeiten).

- Die Methode `equals` wird vom Compiler so generiert, dass sie für inhaltlich gleiche Objekte immer `true` ergibt.
- Der Gleicheitsoperator == kann nur für primitive Typen aber nicht für Objekte verwendet werden.
- Für Objekte ist der Gleicheitsoperator == synonym mit der Methode `equals`.
- Die Methode `equals` wird von der Klasse `Object` geerbt.
- Die Parameter der Methode `equals` ist vom Typ `Object`.

Korrekte Antworten

- Die Methode `equals` wird von der Klasse `Object` geerbt.
- Die Parameter der Methode `equals` ist vom Typ `Object`.

Erläuterung

Der Operator == testet für Objekte die Objektidentität also ob es sich um dasselbe Objekt an der selben Speicheradresse handelt. Will man einen inhaltlichen Gleichheitscheck, so ist die Methode `equals` zu verwenden. Diese ist für Klassen so zu implementieren, dass sie eine Äquivalenzrelation realisiert. Insbesondere String-Objekte sollten unbedingt mit der Methode `equals` verglichen werden und nicht mit dem Operator ==.

Frage: Test auf Klassenzugehörigkeit

Mit welchem Infix-Operator können Sie in Java testen, ob ein Objekt Instanz einer bestimmten Klasse oder Schnittstelle ist?

Korrekte Antworten

- instanceof

Erläuterung

`instanceof` ist tatsächlich als ein Infic-Operator zu betrachten. Es stellt somit eine absolute Sonderrolle in der Programmiersprache Java da.

Frage: Überschreiben von Methoden

Welche der folgenden Aussagen sind bezüglich des Überschreinbens einer Methode in der Programmiersprache Java korrekt?

(Mehrere Antwortmöglichkeiten).

- Es muss die Annotation `@Override` gesetzt werden.
- Der Rückgabetyp muss identisch mit dem der überschriebenen Methode.
- Die Sichtbarkeit muss identisch sein mit der Sichtbarkeit der überschriebenen Methode.
- Der Rückgabetyp darf spezieller sein als in der überschriebenen Methode.
- Die Sichtbarkeit darf nicht geringer sein, als bei der überschriebenen Methode.

Korrekte Antworten

- Der Rückgabetyp darf spezieller sein als in der überschriebenen Methode.
- Die Sichtbarkeit darf nicht geringer sein, als bei der überschriebenen Methode.

Erläuterung

Als Faustregel kann man sagen: beim Überschreiben darf eine Methode keine Eigenschaft der überschriebenen Methode verletzten, falls die neue überschreibene Methode statt dieser aufgerufen wird. Die Annation `@Override` sollte man setzen, damit der Compiler überprüft, ob man tatsächlich eine Methode überschreibt. Sie muss aber nicht gesetzt werden (hauptsächlich nicht, um Kompatibilität zu Java-Code der Version vor Java 1.5 zu gewährleisten).

Frage: Pakete

Welche der folgenden Aussagen sind korrekt in der Programmiersprache Java?

(Mehrere Antwortmöglichkeiten).

- Um eine Klasse zu verwenden, die nicht im Standardpaket liegt, muss diese importiert werden.
- Zu viele Importanweisungen machen das Programm in der Ausführung langsam.
- Importanweisungen werden während der Laufzeit ausgewertet.
- Die Importanweisung

  ```
  import de.uni_xy.*;
  ```

  importiert alle Klasse des Pakets `de.uni_xy` und zusätzlich alle Klassen in Unterpaketen von `de.uni_xy`.
- Importanweisungen werden nur vom Compiler verwendet, um für Klassennamen den vollqualifizierten Namen aufzulösen.
- Das Standardpaket `java.lang` braucht nicht importiert zu werden.

Korrekte Antworten

- Importanweisungen werden nur vom Compiler verwendet, um für Klassennamen den vollqualifizierten Namen aufzulösen.
- Das Standardpaket `java.lang` braucht nicht importiert zu werden.

Erläuterung

Importanweisungen sind lediglich eine Hilfe, um nicht für alle verwendeten Klassen, den vollqualifizierten Klassennamen schreiben zu müssen. Der Compiler verwendet die Importanweisungen, um Klassennamen eindeutig zuzuordnen. Importanweisungen haben keinen Einfluss auf die Laufzeit. Man kann auch Klassen, die nicht imprortiert wurden, verwenden. Hierzu ist einfach ihr vollqualifizierter Name anzugeben.

Frage: Anonyme Klassen

Was bezeichnet man in Java als anonyme Klassen?

(Eine Antwortmöglichkeit.)

- Eine Klasse, die eine Schnitstelle nicht vollständig implementiert.
- Eine Klasse ohne Paketangabe.
- Eine Klasse, die in der Swing-Bibliothek einen *event listener* implementiert.
- Eine Klasse aus einem unbekannten Paket.
- Eine Klasse ohne Namen, von der genau ein Objekt mit `new` erzeugt wird und dabei die abstrakten Methoden direkt implementiert werden.

Korrekte Antworten

- Eine Klasse ohne Namen, von der genau ein Objekt mit `new` erzeugt wird und dabei die abstrakten Methoden direkt implementiert werden.

Erläuterung

Anonyme Klasse sind ein syntaktisches Konstrukt, bei dem direkt in einem Aufruf zu Erzeugung eines Objektes mit dem Schlüsselwort `new`, Methoden für dieses Objekt definiert werden können. Somit hat eine anonyme Klasse genau ein Objekt.

Frage: Für-alle-Schleife (for-each)

Für welche Objekte kann die Für-alle-Schleife zum Iterieren verwendet werden?

(Mehrere Antwortmöglichkeiten).

- Beliebige String-Objekte
- Objekte von Aufzählungsklassen (enum)
- Ganzzahlige Werte
- Alle Reihungen (arrays)
- Objekte, die die Schnittstelle `Iterable` implementieren
- Objekte, die die Schnittstelle `Iterator` implementieren

Korrekte Antworten

- Alle Reihungen (arrays)
- Objekte, die die Schnittstelle `Iterable` implementieren

Erläuterung

Der Compiler übersetzt die Für-alle-Schleife in eine einfache for-Schleife, in der das Iterator-Objekt zum Iterieren verwendet wird. Eine Klasse, die `Iterable` implementiert, hat ein Iteratorobjekt. Reihungen stellen ein Sonderfall da. Auch diese können mit der Für-alle-Schleife iteriert werden.

Frage: Funktionale Schnittstelle

Was bewirkt, die Annotation `@FunctionalInterface` bei Schnittstellen?

(Eine Antwortmöglichkeit.)

- Der Compiler gibt einen Fehler, wenn die Schnittstellen mehr als eine abstrakte Methode hat.
- Nur Schnittstellen mit dieser Annotation können mit einem Lambda-Ausdruck implementiert werden.
- Zur Laufzeit wird überprüft, ob alle Funktionen der Schnittstelle implementiert sind.
- Zusätzliche funktionale Eigenschaften werden durch den Compiler generiert.

Korrekte Antworten

- Der Compiler gibt einen Fehler, wenn die Schnittstellen mehr als eine abstrakte Methode hat.

Erläuterung

Eine Schnittstelle mit nur einer abstrakten Methode wird als funktionale Schnittstelle bezeichnet. Die Annotation bringt den Compiler dazu, diese Eigenschaft zu überprüfen. Funktionale Schnittstellen können mit Lambda-Ausdrücken implementiert werden. Dieses geht aber auch, wenn die Annotation nicht gesetzt wurde.

Frage: Unterklassen

Welche Ausgabe macht folgendes Programm auf der Kommandozeile?

```
class Person{
  String name;
  Person(String name){this.name=name;}
  String anrede(){return "Sehr gehhrte(r) "+name;}}
class Student extends Person{
  int matrNr;
  Student(String name,int matrNr){ super(name);
    this.matrNr = matrNr;
  }
  String anrede(){return "Ey, Alte(r)!";}
  public static void main(String[] args){
    Student s = new Student("Martin Müller",567567);
    Person p = s;
    System.out.println(p.anrede());
  }}
```

(Eine Antwortmöglichkeit.)

- Ey, Alte(r)!
- Sehr gehhrte(r) Martin Müller
- Sehr gehhrte(r) Martin Müller, Alte(r)!

Korrekte Antworten

- Ey, Alte(r)!

Erläuterung

Durch die Späte-Bindung für bei einer überschriebenen Methode immer die Methode aufgerufen, die für die Klasse, von der das entsprechende Objekt erzeugt wird, als letztes definiert wurde.

Frage: Gleichheit

Welche Ausgabe macht folgendes Programm auf der Kommandozeile?

```
class Eq{
  public static void main(String[] args){
    String s1 = "hallo".toUpperCase();
    String s2 = "haLLo".toUpperCase();
    System.out.println(s1==s2);
  }
}
```

(Eine Antwortmöglichkeit.)

- true
- false
- Das hängt vom Locale des Rechners ab.

Korrekte Antworten

- false

Erläuterung

Der Gleichheitsoperator für Objekte in Java prüft nicht eine inhaltliche Gleichheit, die in einer Methode `equals` definiert ist, sondern eine Objektidentität. Es wird auf das identische Objekt und nicht auf ein gleiches Objekt geprüft.

Frage: Konstruktoren

Java kennt die Schlüsselwörter `this` und `super` gefolgt von einer Parameterliste. Welche Aussagen sind über Aufrufe dieser beiden Schlüsselwörter mit einer Parameterliste korrekt?

(Mehrere Antwortmöglichkeiten).

- Der Aufruf von `this` mit einer Parameterliste darf nur direkt nach dem Aufruf von `super` erfolgen.
- Beide Aufrufe dürfen nur als erste Anweisung in einem Konstuktor erfolgen.
- Ein Konstruktor muss als erste Anweisung einen Aufruf von `super` mit Parameterliste haben.
- Die erste Anweisung in einem Konstuktor muss eine der beiden Anweisungsformen sein.
- Der Aufruf von `this` mit Parameterliste kann in jeder Methode erfolgen.

Korrekte Antworten

- Beide Aufrufe dürfen nur als erste Anweisung in einem Konstuktor erfolgen.
- Die erste Anweisung in einem Konstuktor muss eine der beiden Anweisungsformen sein.

Erläuterung

`this` und `super` mit einer Parameterliste rufen einen Konstruktor auf. Dieses darf nur als erste Anweisung innerhalb eines Konstruktors geschehen und die erste Anweisung muss einer dieser Aufrufe sein.

Frage: Überschreiben von Methoden

Wenn in einer Klasse eine Methode überschrieben wird, so kann es manchmal sinnvoll sein, auf die überschriebene Version zuzugreifen. Welche Aussagen sind korrekt, wenn die Methode `m()` aus einer Klasse `Upper` überschrieben wurde.

(Mehrere Antwortmöglichkeiten).

- Auf die überschriebene Methode kann mit dem Name der Klasse, in der diese definiert wurde, zugegriffen werden: `Upper.m()`.
- Auf überschriebene Version kann mit dem Schlüsselwort `super` zugegriffen werden: `super.m()`.
- Ein Aufruf von von `super.super.m()` ermöglicht auch auf eine noch höher in der Vererbung überschriebene Methode zuzugreifen..
- der Aufruf von `super.m()` kann in jeder Methode der Klasse erfolgen.
- Der Aufruf von `super.m()` kann nur in der überschreibenden Version von `m()` erfolgen.

Korrekte Antworten

- Auf überschriebene Version kann mit dem Schlüsselwort `super` zugegriffen werden: `super.m()`.
- der Aufruf von `super.m()` kann in jeder Methode der Klasse erfolgen.

Erläuterung

Man hat mit dem Schlüsselwort `super` nur Zugriff auf die direkt überschriebene Version der Methode.

Frage: Annotation @Override

Was ist bezüglich der Annotation `@Override` korrekt?

(Mehrere Antwortmöglichkeiten).

- Methoden die eine andere Methode überschreiben müssen diese Annotation gesetzt haben.
- Wenn eine Methode überschrieben wird, dann muss dies Annotation gesetzt werden, damit die späte Bindung (late binding) für die Methode funktioniert.
- Sie bringt den Compiler dazu, zu überprüfen, ob die Methode eine geerbte Methode überschreibt.
- Es ist *nicht* zwingend notwendig, dass die Annotation gesetzt wird.

Korrekte Antworten

- Sie bringt den Compiler dazu, zu überprüfen, ob die Methode eine geerbte Methode überschreibt.
- Es ist *nicht* zwingend notwendig, dass die Annotation gesetzt wird.

Erläuterung

Wenn man eine Methode überschreibt, so ist zu empfehlen, die Annotation zu setzen, aber aus Kompatibilitätsgründen ist es nicht notwendig, die Annotation zu setzen.

Frage: Vererbungsende

Mit welchem Schlüsselwort kann verboten werden, dass von einer Klasse Unterklassen definiert werden?

Korrekte Antworten

- final

Erläuterung

Eine populäre Klassen, die `final` ist, ist zum Beispiel die Klasse `String`.