JAVA 05: Objektorientierte Konzepte

panitz

Zusammenfassung

Dieser Kurs beschäftigt sich mit Grundkonzepten der objektorientierten Programmierung. Was sind Klassen und Objekte? Wie sehen Konstruktoren aus? Was bedeutet es Unterklasse zu definieren? Was ist späte Bindung. Wie spielen die Sichtbarkeitsattribute eine Rolle.

Frage: Begriffe (1)

Was repräsentiert eine Entität der realen Welt, welches klar und eindeutig unterschieden und identifiziert werden kann?

(Eine Antwortmöglichkeit.)

- Eine Klasse.
- Ein Objekt.
- Eine Methode.
- Ein Feld.

Korrekte Antworten

- Ein Objekt.

Erläuterung

Objekte Instanzen von Klassen. In der Klasse ist definiert, welche Felder und Methoden die Objekte der Klasse haben.

Frage: Begriffe (2)

Wie benennen wir die Vorlage, die Schablone (template), den Bauplan (blueprint) oder den Vertrag (contract), welcher/s ein Objekt definiert?

(Eine Antwortmöglichkeit.)

- Eine Klasse *(class)*.
- Ein Objekt *(object)*.
- Eine Methode *(method)*.
- Ein Feld *(field)*.

Korrekte Antworten

- Eine Klasse *(class)*.

Erläuterung

Klassen beschreiben eine Menge von Objekten mit bestimmten Eigenschaften. Eine Klasse ist auch der Typ eines Objektes.

Frage: Begriffe (4)

Welche(s) Keyword(s)s benutzt man, um eine Klasse zu definieren?

(Mehrere Antwortmöglichkeiten).

- `interface`
- `class`
- `return`
- `main`

Korrekte Antworten

- `class`

Erläuterung

Es gibt noch eine besondere Art von Klassen, die Aufzählungsklassen, die mit dem Schlüsselwort `enum` eingeleitet werden.

Frage: Begriffe (6)

Welche der folgenden Aussagen sind richtig? Kreuzen Sie die wahre(n) Aussage(n) an.

(Mehrere Antwortmöglichkeiten).

- Instanzen werden in Java bei der ersten Wertzuweisung zu einer Instanzvariablen dieses Objektes erzeugt.
- Mit dem Schlüsselwort `this` kann immer eine Referenz auf die aktuelle Instanz angesprochen werden.
- Klassenvariablen, die nicht an Objekte gebunden sind, werden durch das Schlüsselwort `static` gekennzeichnet.
- Konstruktoren können in Java genau so wie andere Methoden übeschrieben werden.
- Die Methode `toString` kann wie jede andere Methode überschrieben werden, jedoch ist dieses im allgemeinen nicht ratsam.

Korrekte Antworten

- Mit dem Schlüsselwort `this` kann immer eine Referenz auf die aktuelle Instanz angesprochen werden.
- Klassenvariablen, die nicht an Objekte gebunden sind, werden durch das Schlüsselwort `static` gekennzeichnet.

Erläuterung

Mit der Markierung `static` verlässt man sozusagen den objektorientierten Bereich. Eigenschaften die `static` sind, sind nicht mehr individuell für jedes Objekt vorhanden.

Frage: Codeanalyse (1)

Analysieren Sie den folgenden Code:

```
class A{
  String s = "Welcome";
  public String toString(A a){
    return a.s;
  }
  public static void main(String[] args){
    A a = new A();
    System.out.println(a.toString());
  }
}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil die Klasse A keinen Konstruktor hat.
- Das Programm kompiliert nicht, weil die Klasse A die `toString`-Methode mit der Signatur `public String toString()` haben sollte.
- Das Programm kompiliert nicht, weil die Klasse A die `toString`-Methode mit der Signatur `public String toString(String s)` haben sollte.
- Das Programm kompiliert und druckt `Welcome` auf der Kommandozeile aus.
- Das Programm kompiliert und druckt eine Ausgabe der Art `A@210366b4` auf der Kommandozeile aus.

Korrekte Antworten

- Das Programm kompiliert und druckt eine Ausgabe der Art `A@210366b4` auf der Kommandozeile aus.

Erläuterung

Die Methode `toString` der Klasse `Object` wurde nicht überschrieben. Es wurde eine überladene Version von `toString` für die Klasse `A` definiert, die einen Parameter vom Typ `A` hat. Somit wird die Methode `toString` aus der Klasse `Object` aufgerufen.

Frage: Codeanalyse (2)

Analysieren Sie den folgenden Code:

```
class A{
  String s = "Welcome";
  A(String s){
    this.s = s;
  }
  public static void main(String[] args){
    A a = new A();
    System.out.println(a.toString());
  }
}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil die Klasse A keinen Konstruktor hat.
- Das Programm kompiliert nicht, weil die `toString`-Methode nicht in Klasse `A` definiert ist.
- Das Programm kompiliert und druckt `Welcome` auf der Kommandozeile aus.
- Das Programm kompiliert nicht, weil der Konstruktor ohne Argument aufgerufen wird.

Korrekte Antworten

- Das Programm kompiliert nicht, weil der Konstruktor ohne Argument aufgerufen wird.

Erläuterung

Es wurde ein Konstruktor definiert. Damit steht der Standardkonstruktor ohne Parameter, der sonst automatisch generiert wird, für die Klasse `A` nicht generiert.

Frage: Codeanalyse (3)

Analysieren Sie den folgenden Code:

```
class A{
  String s = "Welcome";
  A(String s){
    this.s = s;
  }
  public String toString(){return s;}
  public static void main(String[] args){
    A a = new A("Hello");
    System.out.println(a.toString());
  }
}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil die Klasse A keinen Konstruktor hat.
- Das Programm kompiliert nicht, weil die `toString`-Methode nicht in Klasse `A` definiert ist.
- Das Programm kompiliert und druckt `Welcome` aus.
- Das Programm kompiliert und druckt `Hello` aus.
- Das Programm kompiliert nicht, weil der Konstruktor mit falschem Argument aufgerufen wird.

Korrekte Antworten

- Das Programm kompiliert und druckt `Hello` aus.

Erläuterung

Es wird ein `A`-Objekt mit dem String »Hello« erzeugt. Für dieses wird die `toString`-Methode aufgerufen. Diese ist in der Klasse `A` überschrieben worden und gibt den im Konstruktor initialisierten Wert des Feldes `s` zurück.

Frage: Codeanalyse (4)

Analysieren Sie den folgenden Code:

```
class A{
  private int x = 0;
  private int _y = 42;
  A(){ this.x = 1; this._y = 1;}
  int getY(){return _y;}
  public static void main(String[] args){
    A a = new A();
    System.out.println(a.x);
  }}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil x private ist und auf x nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil y private ist und auf y nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil man eine Variable nicht _y nennen darf.
- Das Programm kompiliert und druckt 1 aus.
- Das Programm kompiliert und druckt 0 aus.

Korrekte Antworten

- Das Programm kompiliert und druckt 1 aus.

Erläuterung

Frage: Codeanalyse (5)

Analysieren Sie den folgenden Code:

```
class A{
  int x = 0;
  private int _y = 42;
  A(){this.x = 1; this._y = 1;}
  int getY(){return _y;}
}
class B{
  public static void main(String[] args){
    A a = new A();
    System.out.println(a._y);
  }}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil x private ist und auf x nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil $_y private ist und auf _y nicht von außerhalb der Klasse zugegriffen werden kann.$
- Das Programm kompiliert nicht, weil man eine Variable nicht _y nennen darf.
- Das Programm kompiliert und druckt 1 aus.
- Das Programm kompiliert und druckt 0 aus.

Korrekte Antworten

- Das Programm kompiliert nicht, weil $_y privateistundauf_y nichtvonauerhalbderKlassezugegriffenwerdenkann.$

Erläuterung

Private Felder dürfen nur in der eigenen Klasse angesprochen werden.

Frage: Codeanalyse (6)

Analysieren Sie den folgenden Code:

```
class A{
  int x; private int _y;
  int getY(){return _y;}
}
class B{
  public static void main(String[] args){
    A a = new A();
    a.x = 45;
    System.out.println(a.x);
  }}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil x private ist und auf x nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil y private ist und auf y nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil man eine Variable nicht _y nennen darf.
- Das Programm kompiliert und druckt 1 aus.
- Das Programm kompiliert und druckt 45 aus.

Korrekte Antworten

- Das Programm kompiliert und druckt 45 aus.

Erläuterung

Beide Klasse sind im `default`-Paket. Felder ohne Sichtbarkeitsattribut haben die Sichtbarkeit für alle Klassen im gleichen Paket. Deshalb kann man in Klasse `B` auf das Feld `x` aus Klasse `A` zugreifen.

Frage: Codeanalyse (7)

Analysieren Sie den folgenden Code:

```
class A{
  int x; private int y = 1;
  int getY(){return y;}
  void setY(int y){y = y;}
}
class B{
  public static void main(String[] args){
    A a = new A();
    a.setY(45);
    System.out.println(a.getY());
  }}
```

Was gilt für dieses Programm?

(Eine Antwortmöglichkeit.)

- Das Programm kompiliert nicht, weil x private ist und auf x nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil y private ist und auf y nicht von außerhalb der Klasse zugegriffen werden kann.
- Das Programm kompiliert nicht, weil man eine Variable nicht y nennen darf.
- Das Programm kompiliert und druckt 1 aus.
- Das Programm kompiliert und druckt 45 aus.

Korrekte Antworten

- Das Programm kompiliert und druckt 1 aus.

Erläuterung

Unglücklicher Weise ist die *Setter*-Methode der Klasse `A` fehlerhaft implementiert. sie müsste lauten:

```
void setY(int y){
  this.y = y;
}
```

Frage: Codeanalyse (9)

Was druckt dieses Programm aus?

```
class Count{ int count=0;}
class A{
  static int times = 0;
  static Count myCount = new Count();
  static void increment(Count c, int times){
    c.count += 1;
    times += 1;
  }
  public static void main(String[] args){
    for (int i=0;i< 100;i++){
      increment(myCount, times);
    }
    System.out.println("myCount.count="+myCount.count+" times ="+times);
  }}
```

(Eine Antwortmöglichkeit.)

- myCount.count = 101 times = 0
- myCount.count = 100 times = 0
- myCount.count = 100 times = 100
- myCount.count = 101 times = 101

Korrekte Antworten

- `myCount.count = 100 times = 0`

Erläuterung

Der primitive Typ `int` wird für den Parameter `times` per Wert übergeben. Eine Erhöhung des Parameters `times` hat keinen Effekt auf das statische Feld `times` der Klasse `A`. Das Objekt `myCount` wird hingegen per Referenz übergeben. Es kann innerhalb der Methode `increment` verändert werden.

Frage: Codeanalyse (11)

Was druckt dieses Programm aus?

```
class A{
   int i;
   A(){i = 1;}
}
class B extends A{
  int j;
  B(){super(); j = 2;}
  public static void main(String[] args){
    B b = new B();
    System.out.println(b.i+" "+b.j);
  }
}
```

(Eine Antwortmöglichkeit.)

- 0 0
- 0 1
- 0 2
- 1 2
- 2 1

Korrekte Antworten

- `1 2`

Erläuterung

Objekte der Klasse `B` haben zwei Felder: das geerbte Feld `i` und das in `B` deklarierte Feld `j`.

Frage: Codeanalyse (10)

Analysieren Sie den folgenden Code:

```
class A{
  int i = 1;
}
class B extends A{
  int j = 2;

  public static void main(String[] args){
    A b = new B();
    System.out.println(b.i)
    System.out.println(b.j)
  }
}
```

Warum kompiliert das Programm nicht?

(Eine Antwortmöglichkeit.)

- Klasse `B` erbt von `A`, aber das Feld `i` in `A` wurde nicht vererbt.
- Die Variable `b` ist vom Typ `A` und `A` enthält kein Feld `j`.
- Klasse `B` erbt von `A` und damit automatisch auch das Feld `i`, allerdings wird dieses nicht gesetzt.
- Für das Erzeugen von **b** im Konstructor muss ein Argument angeben werden, z.B. `new B(42)`.

Korrekte Antworten

- Die Variable `b` ist vom Typ `A` und `A` enthält kein Feld `j`.

Erläuterung

Woher soll ich bei der Variable vom Typ `A` wissen, dass dort ein Objekt der Klasse `B` drin gespeichert ist.

Frage: Codeanalyse (12)

Was druckt dieses Programm aus?

```
class ParentClass{
  int x = 1; int y = 10;
  public String toString(){
    return x+" "+y;
  }}
class ChildClass extends ParentClass{
  int x;
  ChildClass(){
    this.x = 2;
    this.y = 20;
  }
  public static void main(String[] args){
    ParentClass c = new ChildClass();
    System.out.println(c);
  }}
```

(Eine Antwortmöglichkeit.)

- 1 10
- 1 20
- 2 10
- 2 20

Korrekte Antworten

- 1 20

Erläuterung

Bei Feldern gibt es anders als bei Methoden keine späte Bindung. Es existieren zwei unterschiedliche Felder `x` und der Typ der Variable, von der das Feld `x` angesprochen wird, entscheidet, auf welches `x` zugegriffen wird.

Frage: Codeanalyse (13)

Es sei `A` eine Unterklasse von `B`. Was schreibt man, um einen Konstruktor von B von A aus aufzurufen?

(Eine Antwortmöglichkeit.)

- super()
- A.super()
- A()
- this.A()

Korrekte Antworten

- super()

Erläuterung

Und dieses darf nur die erste Anweisung in einem Konstruktor sein und sonst nirgendwo aufgerufen werden.

Frage: Codeanalyse (15) - Method Overriding

Was wird durch folgenden Code ausgegeben?

```
class A{
  int i = 0;
  A(int i){ this.i = i;}
  void m1(){this.i += 1;}
}
class B extends A{
  int j = 0;
  B(){super(3);}
  void m1(){ this.j += 1;}
  public static void main(String[] args){
    B b = new B();
    b.m1();
    System.out.println(b.i+" "+b.j);
  }}
```

(Eine Antwortmöglichkeit.)

- 2 0
- 3 1
- 4 0
- 3 0
- 4 1

Korrekte Antworten

- `3 1`

Erläuterung

Durch die späte Bindung (Late Binding) wird die Methode `m1` der Klasse `B` aufgerufen. Diese erhöht das Feld `j`. Das Feld `i` wird somit nicht verändert.

Frage: Codeanalyse (16) - Method Overriding

Welche der folgenden Aussagen sind wahr?

(Eine Antwortmöglichkeit.)

- Beim Ausführen des Konstruktors der Unterklasse, wird auch immer ein Konstruktor der Superklasse ausgeführt.
- Eine non-private Methode der Superklasse kann überschrieben (`@Override`) werden.
- Man kann auch die Konstruktoren überschreiben.
- Man kann alle Methoden der Superklasse überschreiben, insbesondere auch private Methoden.

Korrekte Antworten

- Beim Ausführen des Konstruktors der Unterklasse, wird auch immer ein Konstruktor der Superklasse ausgeführt.
- Eine non-private Methode der Superklasse kann überschrieben (`@Override`) werden.

Erläuterung

Private Methoden sind ja selbst in Unterklassen nicht sichtbar.

Frage: Die Klasse Object (4)

Welche der folgenden Aussagen sind wahr?

(Mehrere Antwortmöglichkeiten).

- Ein Konstruktor `Object()` ist in der Klasse `Object` vorhanden.
- Die Methode `public String toString()` ist in der Klasse `Object` definiert.
- Die Methode `public equals (Object other)` ist in der Klasse `Object` definiert.
- Die Methode `public long getID()` ist in der Klasse `Object` definiert.
- Die Methode `public Code getCode()` ist in der Klasse `Object` definiert.

Korrekte Antworten

- Ein Konstruktor `Object()` ist in der Klasse `Object` vorhanden.
- Die Methode `public String toString()` ist in der Klasse `Object` definiert.
- Die Methode `public equals (Object other)` ist in der Klasse `Object` definiert.

Erläuterung

Es gibt hingegen eine Methode `getClass()` in der Klasse `Object`.

Frage: Codeanalyse (17)

Was wird durch folgenden Code ausgegeben?

```
class A{
  protected int i =1;
  int m(){ return this.i = 10;}
}
class B extends A{
  int m(){return this.i += 1;}
  public static void main(String[] args){
    A b = new B();
    System.out.println(b.m());
  }
}
```

(Eine Antwortmöglichkeit.)

- 1
- 2
- 10
- 11
- Das Programm kompiliert nicht, weil i von b aus nicht zugreifbar ist.

Korrekte Antworten

- 2

Erläuterung

Wegen der späten Bindung wird die Methode `m` aus der Klasse `B` ausgeführt, obwohl die Variable `b` für den Typ der Klasse `A` deklariert wurde.

Frage: Codeanalyse (18)

Was wird durch folgenden Code ausgegeben?

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

(Eine Antwortmöglichkeit.)

- A A A
- B B B
- B A C
- A B C

Korrekte Antworten

- `A B C`

Erläuterung

Jede Klasse hat eine eigene überschrieben Methode `toString`. Jedes Objekt führt die Methode `toString` der Klasse aus, von der das Objekt mit `new` erzeugt wurde.

Frage: Codeanalyse (19)

Was wird durch folgenden Code ausgegeben?

```
class A{
  public String toString(){
    return "A";
  }
}
class B extends A{}
class C extends B{
  public static void main(String[] args){
    A b = new B();
    A a = new A();
    A c = new C();
    System.out.println(a+" "+ b+" "+ c);
  }
}
```

(Eine Antwortmöglichkeit.)

- A A A
- B B B
- B A C
- A B C

Korrekte Antworten

- `A A A`

Erläuterung

Die Klassen `B` und `C` erben die Methode `toString` von der Klasse `A`. Hier wird der Buchstabe A zurück gegeben.

Frage: Codeanalyse (20)

Was wird durch folgenden Code ausgegeben?

```
class A{
  int i;  int j;
  A(int i,int j){
    this.i=i;
    this.j=j;
  }
  public String toString(){return "A";}
  public boolean equals(Object that){
    return i *j == ((A)that).i * ((A)that).j;
  }
  public static void main(String[] args){
    A x = new A(1, 2);
    A y = new A(2, 1);
    System.out.println(x .equals(y) );
  }}
```

(Eine Antwortmöglichkeit.)

- `true`
- `false`
- 0
- Es kommt zu einer `ClassCastException`.

Korrekte Antworten

- `true`

Erläuterung

Nach der Implementierung sind zu `A` Objekten andere `A` Objekte gleich, wenn jeweils das Produkt der Felder `i` und `j` denselben Wert hat. Wird diese `equals`-Methode mit einem Objekt, das nicht zur Klasse `A` gehört aufgerufen, kommt es allerdings tatsächlich zu einer `ClassCastException`.

Frage: Codeanalyse (23)

Was wird durch folgenden Code ausgegeben?

```
class Person{
  String getInfo(){ return "Person aufgerufen";}
  void printPerson(){
    System.out.print(this.getInfo()+" ");
  }}
class Student extends Person{
  String getInfo(){ return "Student aufgerufen";}
  public static void main(String[] args){
    new Person().printPerson();
    new Student().printPerson();
  }}
```

(Eine Antwortmöglichkeit.)

- `Person aufgerufen Person aufgerufen`
- `Person aufgerufen Student aufgerufen`
- `Student aufgerufen Person aufgerufen`
- `Student aufgerufen Student aufgerufen`
- Es kommt zu einem Laufzeitfehler wegen falscher Initialisierung.

Korrekte Antworten

- `Person aufgerufen Student aufgerufen`

Erläuterung

Durch die späte Bindung wird für Studenten-Objekte bei `getInfo` auch die in der Klasse `Student` überschriebene Version aufgerufen.

Frage: Codeanalyse (24)

Was wird durch folgenden Code ausgegeben?

```
class Person{
  private String getInfo(){ return "Person aufgerufen";}
  void printPerson(){
    System.out.print(this.getInfo()+" ");
  }}
class Student extends Person{
  private String getInfo(){ return "Student aufgerufen";}
  public static void main(String[] args){
    new Person().printPerson();
    new Student().printPerson();
  }}
```

(Eine Antwortmöglichkeit.)

- `Person aufgerufen Person aufgerufen`
- `Person aufgerufen Student aufgerufen`
- `Student aufgerufen Person aufgerufen`
- `Student aufgerufen Student aufgerufen`
- Es kommt zu einem Laufzeitfehler wegen falscher Initialisierung.

Korrekte Antworten

- `Person aufgerufen Person aufgerufen`

Erläuterung

Gemeiner Weise ist die Methode `getInfo` auf `private` gesetzt. Damit kann sie nicht richtig überschrieben werden und es funktioniert keine späte Bindung.

Frage: Self-Inspection

Was kann man benutzen, um festzustellen, ob ein Objekt `o` eine Instanz der Klasse `A` ist?

(Mehrere Antwortmöglichkeiten).

- `o.isInstance(A)`
- `o.getClass()==A.class`
- `o instanceof A`
- `o instanceof A.class`

Korrekte Antworten

- `o.getClass()==A.class`
- `o instanceof A`

Erläuterung

Der Unterschied zwischen `instanceof` und `getClass` ist, dass mit ersteren auch automatisch auf eventuelle Unterklasse geprüft wird, bei letzteren auf exakt die Klasse und für Unterklassen von `A` bei `o.getClass()==A.class` bereits falsch zurück gegeben wird.