Beispiel Übungsaufgaben. Es wird keine Garantie auf Stimmigkeit gegeben!

Übungsaufgaben Präsenzübung

Aufgabe 1) Interface und abstrakte Klasse

Welche **drei** Compilerfehler erhalten Sie für folgendes Programm?

```
interface I {
   void m();
}

abstract class A {
      void n();
}

class B extends A implements I {
      void n();
      abstract void k();
}
```

Lösung:

- Die Klasse B muss die Methode m implementieren oder muss abstrakt sein.
- Die Methode n in A muss entweder abstrakt sein oder mit einem Methodenrumpf versehen werden.
- Die Methode k in B darf entweder nicht abstrakt deklariert werden oder B muss abstrakt sein.

Aufgabe 2) Interface vs. Abstrakte Klasse

Nennen Sie **einen** Vorteil abstrakter Klassen gegenüber Interfaces.

Lösung:

Abstrakte Klassen können im Gegensatz zu Interfaces auch nicht-abstrakte Methoden implementieren, also ein Teilverhalten einer Klasse vorgeben.

Nennen Sie **einen** Vorteil von Interfaces gegenüber

Lösung:

Interfaces können in beliebiger Anzahl von einer Klasse implementiert werden. Während eine Klasse also maximal eine Superklasse haben kann, können ihr mit Hilfe von Interfaces darüber hinaus noch beliebig viele weitere direkte Supertypen gegeben werden.

Aufgabe 3) Interface und Abstrakte Klasse Gemeinsamkeiten

Nenne Sie **zwei** Gemeinsamkeiten von abstrakten Klassen und Interfaces.

Lösung:

- Abstrakte Klassen und Interfaces deklarieren Typen.
- Weder eine abstrakte Klasse noch ein Interface lassen sich instanziieren.

Wozu dienen Interfaces?

Lösung:

Durch ein Interface kann eine öffentliche Schnittstelle als Typ definiert werden. Klassen können Interfaces als Supertypen angeben und damit zusagen, dass ihre Instanzen diese Schnittstelle bedienen können.

Wozu dienen abstrakte Klassen?

Lösung:

Abstrakte Klassen dienen dazu, von mehreren Supertypen gemeinsam genutzte Teile der Implementierung an einer Stelle zusammenzufassen und Programme dadurch pflegeleichter zu machen.

Wieso erlaubt der Java-Compiler es nicht, eine Methode einer abstrakten Klasse sowohl abstract als auch private zu deklarieren?

Lösung:

Eine Methode, welche abstract deklariert ist, verfügt über keine Implementierung. Durch ihre Deklara- tion wird gewissermaßen bekannt gegeben, dass eine Methode mit dieser Signatur auf dem Objekt aufgerufen werden kann. Welcher Code beim Aufruf ausgeführt werden soll, ist jedoch nicht definiert und muss erst in den Subklassen festgelegt werden. Dies geschieht, indem die abstrakte Methodendeklarati- on überschrieben und eine Implementierung angegeben wird.

Methoden, welche private deklariert sind, sind jedoch in den Subklassen nicht sichtbar, sodass sie ins- besondere auch nicht zum Überschreiben zur Verfügung stehen. Für abstract deklarierten Methoden mit private

Sichtbarkeit könnte so niemals eine Implementierung angegeben werden und die abstrakte Klasse wäre nutzlos.

Aufgabe 4) Vererbung Methoden überschreiben

Gegeben sei folgendes Programm. Welche Ausgabe erzeugt es?

```
public class Test {
  public static void main(String[] args)
  {

       Object o1 = new Object();
       Object o2 = new String();
       Double d  = new Double();
       String s  = new String();

       A a = new B();

       a.m(o1);
       a.m(o2);
       a.m(d);
       a.m(s);

    }
}
```

```
class A {
   void m(String s) {
      System.out.println(„A.m(String s“);
   }
   void m (Object o) {
      System.out.println(„A.m(Object o)“);
   }
   private void m (Double d) {
      System.out.println(„A.m(Double d)“);
   }
}

class B extends A {
   void m(String s) {
      System.out.println(„B.m(Strings)“);
   }
   void m(Object o) {
      System.out.println(„B.m(Object o)“);
   }
}
```

Lösung:

- `a.m(o1)` `//B.m(Object o)`

Der erste Aufruf von m übergibt ein als Object typisierten Parameter an eine Methode m in A und bindet somit zunächst an A.m(Object). Zur Laufzeit ist der Empfänger des Methodenaufrufs vom Typ B, so dass an die überschriebene Methode B.m(Object o) gebunden wird.

- `a.m(o2)` `//B.m(Object o)`

Der zweite Aufruf von m übergibt ebenso ein als Object typisierten Parameter an eine Methode m in A und bindet somit auch zunächst an A.m(Object). Dass zur Laufzeit ein Object vom Typ String übergeben wird, entzieht sich der Kenntnis des Compilers, so dass nicht die überladene Methode A.m(String) gebunden wird. Zur Laufzeit ist der Empfänger des Methodenaufrufs vom Typ B, so dass an die überschriebene Methode B.m(Object o) gebunden wird.

- `a.m(d)  //B.m(Object o)`

Der dritte Aufruf von m übergibt ein als Double typisierten Parameter an eine Methode m in A. Eine solche Methode A.m(Double) existiert zwar, allerdings ist sie (da private) nicht zugreifbar, weswegen an die einzige weitere passende überladene Methode m(Object) gebunden wird. Zur Laufzeit ist der Empfänger des Methodenaufrufs vom Typ B, so dass an die überschriebene Methode B.m(Object o) gebunden wird.

- `a.m(s)  //B.m(Strings)`

Der vierte Aufruf von m übergibt ein als String typisierten Parameter an eine Methode m in A und bindet somit zunächst an A.m(String).
Zur Laufzeit ist der Empfänger des Methodenaufrufs vom Typ B, so dass an die überschriebene Methode B.m(String s) gebunden wird.

Sichtbarkeiten kurze Übersicht

```
public class A {
    public      int pub;
    protected   int pro;
    private     int piv;
    /*default*/ int def;   // = package private

                /* 0 */
}
```

```
public class B extends A {
    A a = new A();

            /* 1 */
}
```

```
public class C {
    A a = new A();
        /* 2 */
}
```

- An der Stelle /* 0 */

Kann auf alle Variablen zugegriffen werden.

- An der Stelle /* 1 */

gilt folgendes:

Falls A und B im gleichen Paket sind, sind public, protected und default sichtbar
Falls A und B nicht im gleichen Paket sind, sind public und protected sichtbar

- An der Stelle /* 2 */

gilt folgendes:

Falls A und B im gleichen Paket sind, sind public protected und default sichtbar
Falls A und B nicht im gleichen Paket sind, ist nur public sichtbar

Aufgabe 5) Sichtbarkeiten

Gegeben sind vier Java-Klassen in den beiden Dateien A.java und C.java.
Untersuchen Sie welche der Programmzeilen a) - t) gültige Zuweisungen enthalten.

```
package de.tum.ws2009.endterm.source.one;

public class A {
  private static int a1;
  public int a2;
  int a3;
  protected int a4;

  void m() {
    A.a1 = 1;        // a)
    a2 = 2;          // b)
    a3 = 3;          // c)
    a4 = 4;          // d)
    B.b1 = 5;        // e)
  }
}

class B {
  public static int b1;

  void m(A a) {
    A.a1 = 6;        // f)
    a.a2 = 7;        // g)
    a.a3 = 8;        // h)
    a.a4 = 9;        // i)
    B.b1 = 10;       // j)
  }
}
```

a)	☒ Richtig	○ Falsch
b)	☒ Richtig	○ Falsch
c)	☒ Richtig	○ Falsch
d)	☒ Richtig	○ Falsch
e)	☒ Richtig	○ Falsch
f)	○ Richtig	☒ Falsch
g)	☒ Richtig	○ Falsch
h)	☒ Richtig	○ Falsch
i)	☒ Richtig	○ Falsch
j)	☒ Richtig	○ Falsch

```
package de.tum.ws2009.endterm.source.two;

import de.tum.ws2009.endterm.source.one.A;

public class C extends A {
  protected static int c1;

  void m(A a) {
    A.a1 = 11;        // k)      ○ Richtig   ✗ Falsch

    a2 = 12;          // l)      ✗ Richtig   ○ Falsch

    a.a3 = 13;        // m)      ○ Richtig   ✗ Falsch

    this.a4 = 14;     // n)      ✗ Richtig   ○ Falsch

    B.b1 = 15;        // o)      ○ Richtig   ✗ Falsch
  }
}

class D {
  int d1;

  void m(A a) {
    A.a1 = 16;        // p)      ○ Richtig   ✗ Falsch

    a.a2 = 17;        // q)      ✗ Richtig   ○ Falsch

    a.a3 = 18;        // r)      ○ Richtig   ✗ Falsch

    a.a4 = 19;        // s)      ○ Richtig   ✗ Falsch

    C.c1 = 20;        // t)      ✗ Richtig   ○ Falsch
  }
}
```

Aufgabe 6) Überschreiben und Überladen

Betrachten Sie die Methode bar des nachfolgenden Java-Programms und notieren Sie an den Stellen a) bis i) welchen Wert die Variable var nach der Ausfu¨hrung der jeweiligen Programmzeile hat.

```
class A {
  int x = 1;

  int func() {   return - x;   }

  int getX() {    return x;   }
}

public class B extends A {
  int x = 2;

  int func() {   return x * x;   }

  void setInt(int a) {   a = 11;   }

  void setInt(A a) {   a.x = 13;   }

  void bar() {

    int var = 0;

    var = this.x;            // a)  var == __2__

    var = super.x;           // b)  var == __1__

    var = func();            // c)  var == __4__

    var = super.func();      // d)  var == _-1__

    A a1 = (A)this;

    var = a1.func();         // e)  var == __4__

    var = getX();            // f)  var == __1__

    setInt(var);             // g)  var == __1__ (Wert unveraendert zu f))

    setInt(a1);
    var = a1.x;              // h)  var == _13__

    A a2 = new B();
    var = a2.func();         // i)  var == __4__
  }

  public static void main(String[] args) {
    B b = new B();
    b.bar();
  }
}
```

Aufgabe 7) Polymorphie

Welche Arten von Polymorphie kennen Sie?

Lösung:

- Subtyp-Polymorphie
- parametrische Polymorphie
- beschränkt parametrische Polymorphie
- Ad-hoc Polymorphie

Charakterisieren Sie jede Art kurz

Lösung:

- Subtyp-Polymorphie

wird durch Subtyping erreicht. Das bedeuted, dass an den Stellen, an denen ein Objekt vom Typ T erwartet wird, auch Objekte beliebiger Typen S stehen dürfen, sofern diese Typen von T sind.

- parametrische Polymorphie

wird dadurch realisiert, dass bestimmte Konstrukte einer Programmier-sprache - z.B. Methoden oder Klassen - durch Typparameter parametrisiert werden. Bei der Verwendung eines so parametrisierten Konstrukts wird der Typparameter durch einen konkreten Typen ersetzt.

- beschränkt parametrische Polymorphie

entsteht gewissermaßen aus der Kombination von Subtyp-Polymorophie und parametrischer Polymorphie. Parameter können durch Angabe eines minimalen Supertypen T beschränkt werden. Dadurch ist innerhalb der Implementierung des mit der Typvariablen parametrisierten Konstrukts sichergestellt, dass die Typvariable mit T oder einem Subtyp belegt ist. So kann auch ohne Typkonvertierung (Type Cast) innerhalb der generischen Implementierung auf die durch T deklarierte Schnittstelle zugegriffen werden.

- Ad-hoc Polymorphie

bezeichnet das Überladen von Operatoren bzw. Methodennamen. Für identifizierte Operationen können so für unterschiedliche Argumenttypen jeweils optimierte Implementierungen angegeben werden.

Aufgabe 8) Kontrollfluss und der this-Parameter

Gegeben sei folgendes Java-Programm:

```
public class EineKlasse {
	public static void main(String[] args){
		new EineKlasse();
	}
	int zahl;
	EineKlasse() {
		this(32);
	}
	EineKlasse(int zahle) {
		System.out.println(zahl);
		this.zahl = 64;
		methode(new InnereKlasse(128), 256));
		WeitereInnereKlasse.methode(new SubInnereKlasse(4096));
	}
	void methode(InnereKlasse innereKlasse, int zahl) {
		System.out.println(EineKlasse.this.zahl);
	}
	class InnereKlasse{
		int zahl = 512;
		InnereKlasse(int zahl) {
			System.out.println(this.zahl);
		}
```

```
    }
    class SubInnereKlasse extends InnereKlasse {
        int zahl = 1024;
        public SubInnereKlasse(int zahl) {
            super(2048);
            System.out.pritnln(zahl);
        }
    }
    static class WeitereInnereKlasse {
        {    /* 2 */
        }
        static void methode(InnereKlasse innereKlasse) {
            System.out.println(innereKlasse.zahl);
        }
    }
    static void andereMethode() {
        /* 1 */
    }
```

Lösung:

32

512

64

512

4096

512

An der mit /* 1 */ markierten Stelle wird die folgende Anweisung eingefügt. Warum weist der Compiler sie zurück? ***`System.out.println(this.zahl);`***

Lösung:

Statische Methode kann Klassen- aber keine Objektattribute anfragen.

An der mit /* 2*/ markierten Stelle wird die folgende Anweisung eingefügt. Warum weist der Compiler sie zurück? **`System.out.println(EineKlasse.this.zahl);`**

Lösung:

Der Initialisierungsblock, in den diese Anweisung eingefügt werden soll, gehört zu der inneren Klasse WeitereInnereKlasse, welche static deklariert ist. Als solche ist sie – genauso wie auch statische Methoden – nicht einem Objekt der umschließenden Klasse EineKlasse zugeordnet, sondern der Klasse selbst. Eine Instanz der umschließenden Klasse EineKlasse, welche mit EineKlasse.this referenziert werden könnte, existiert somit gar nicht und das Programm ist ungültig.

Aufgabe 9) Vererbung Methoden überschreiben

Wie lautet die Ausgabe des Programms? Begründen Sie Ihre Antwort kurz.

```
public class Test {
	public static void main(String[] args) {

		Super sup = new Sub(); System.out.println(sup.a);
	System.out.println(sup.m());

	}

}

class Super {
	String a = "Super-Attribut"; String m() {

		return "Super-Methode";

	}

}


class Sub extends Super {

	String a = "Sub-Attribut";
	String m() {
	return "Sub-Methode";

	}

}
```

Lösung:

Die Ausgabe dieses Programms lautet:

Super-Attribut
Sub-Methode
Denn: Beim statischen Binden bei Attributen in Java spielt nur der Deklarationstyp des Ausdrucks, über den die Attributselektion erfolgt, eine

Rolle. Der Deklarationstyp der Variablen sup ist Su- per, also wird die Variable a des Typs Super an die Attributselektion gebunden.

Bei der Bindung der Methode durch die VM wird der Laufzeittyp (= dynamischer Typ) des von sup referenzierten Objekts berücksichtigt. Dies ist der Typ Sub. Da die Methode m() aus Super in Sub überschrieben wurde, wird die Methode m() von Sub ausgeführt („dynamisches Binden“).

Aufgabe 10) Zuweisungen und Operatoren

Bei den nachfolgenden Fragen seien jeweils die Variablen wie folgt vorausgesetzt:

```
int a = 2;

int b = 10;

double c = 3.0;

double d = 1.0;

boolean e = false;
```

Berechne die folgenden Ausdrücke und gib für jeden die Reihenfolge an, in der die Operatoren gemäß den Präzedenzen in Java ausgewertet werden. Gib außerdem den Wert und den Datentyp des Ergebnisses an. In der ersten Zeile der Tabelle ist ein Beispiel angegeben.

Ausdruck	1	2	3	4	5	Wert	Datentyp
`z.B.: a = 3 + 4 * 2`							
`3 * 8 / 3 / 2`							
`(int) - (c/ ++d)`							
`c = b % a`							
`a++ - b * c`							
`e = 16 >> 1 >> 2 % 2 == 8`							

```
    public void foo(int n)
    {
        while ((n > 0) && (x-- >= 0))
            ;
        Std.out.println(x);
    }
}
```

```
import eip.*;

public class TestABC
{
    public static void main(String[] args)
    {
        A a = new A(1);
        B b = new B(0);
        C c = new C(5);
        b.foo(50);
        c.foo(50);
        int y = 5;
        b.increment(y);
        Std.out.println(y);
    }
}
```

Aufgabe 10) Vererbung Überschreiben von Methoden

Betrachte die folgenden vier Java-Klassen:

```
import eip.*;

public class A
{
    public int x;

    public A(int x)
    {
        this.x = x;
        Std.out.println("Erzeuge A-Objekt");
    }

    public void foo(int n)
    {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                x++;

        Std.out.println(x);
    }
}
```

```
import eip.*;

public class B extends A
{
    public B(int x)
    {
        super(x);
        Std.out.println("Erzeuge B-Objekt");
    }

    public void increment(int y)
    {
        y++;
    }
}
```

```
import eip.*;

public class C extends B
{
    public C(int x)
    {
        super(x);
        Std.out.println("Erzeuge C-Objekt");
    }
```

Welche Bildschirmausgabe produziert die Ausführung der Klasse TestABC?

Lösung:

//Erzeuge A Objekt

//Erzeuge A Objekt

//Erzeuge B Objekt

//Erzeuge A Objekt

//Erzeuge B Objekt

//Erzeuge C Objekt

//2500 (= 50 * 50)

//4 3 2 1 0 -1

//5

```
package packageA;

    class D extends C {

    }
```

Aufgabe 11) Vererbung und Sichtbarkeit

```
package packageA;

   class A {

      private String ausgabe() {
         return „Das ist Klasse A“;
      }
}
```

```
package packageA;

 package packageA;

    class C extends B {

       public String ausgabe() {
          return „Das ist Klasse C“;
       }
 }
```

```
package nichtpackageA;

   class Test {
      public static void main(String [] args) {
          D d = new D();
          d.ausgabe();
          C c = new C();
          c.ausgabe();
          B b = new B();
          b.ausgabe();
          A a = new A();
          a.ausgabe();
      }
   }
```

Welche Ausgaben erscheinen auf der Konsole bei den vier Klassen und ausgeführter main-Methode?

Lösung:

- d.ausgabe() //Das ist Klasse C
- c.ausgabe() //Das ist Klasse C
- b.ausgabe() //Fehler, da Methode ausgabe() in Klasse B protected
- a.ausgabe() //Fehler, da Methode ausgäbe() in Klasse A private

Aufgabe 12) Vererbung und Sichtbarkeit

Gegeben seien folgende Ausschnitte aus zwei Java-Dateien B.java(links) und A.java(rechts). Mit dem nachfolgenden Java-Hauptprogramm.

```java
package usedemo;
import demo.A;

public class B extends A {

   private int att1 = 5;

   public void hello() {
      System.out.println("B");
      System.out.println(att1);
      System.out.println(att2);
      System.out.println(att3);
      System.out.println(att4);
   }
}
```

```java
package demo;
public class A {

   private    int   att1 = 1;
              int   att2 = 2;
   protected  int   att3 = 3;
   public     int   att4 = 4;

   public void hello(){
      System.out.println("A");
      System.out.println(att1);
      System.out.println(att2);
      System.out.println(att3);
      System.out.println(att4);
   }
}
```

```java
import demo.*;
import usedemo.*;

public class Example {

   public static void main(String[] args) {

      new A().hello();
      new B().hello();

   }
}
```

Beim Übersetzen wird der Compiler zunächst einen oder mehrere Fehler anzeigen. Wo?

Gehen Sie nun davon aus, dass alle fehlerhaften Zeilen auskommentiert sind, und geben Sie die Ausgabe des Programms, d.h. der Methode main an.

Lösung:

kein Zugriff auf das Attribut att2 in der Methode hello() der Klasse B, da dieses in der Super-Klasse A den no-modifier (auch: package-modifier) zugewiesen hat.

Aufgabe 13) Vererbung, Überschreiben, Konstruktoren

```
public class A {

    public int x = 2;

    public A() {
        this.x++;
    }

    public A(int x) {
        this.x += x;
    }

    public void f(double x) {
        this.x = (int) (x + B.y);
    }
}
```

```
public class B extends A {

    public static double y = 3;

    public double x = 0;

    public B(double x) {
        y++;
    }

    public void f(int y) {
        this.x = y * 2;
        B.y = 0;
    }

    public void f(double y) {
        this.x = 2 * y + B.y;
    }
}
```

```
public class M {
    public static void main(String[] args) {
        A a = new A((int) B.y);
        System.out.println(a.x);                // OUT: [    ]

        B b = new B(2);
        System.out.println(b.x + " " + B.y);    // OUT: [    ]    [    ]

        A z = b;
        System.out.println(z.x);                // OUT: [    ]

        z.f(-5.0);
        System.out.println(b.x + " " + z.x);    // OUT: [    ]    [    ]

        z.f(-6);
        System.out.println(b.x + " " + B.y);    // OUT: [    ]    [    ]
    }
}
```

Lösung:

// OUT: [5]

// OUT: [0.0] [4.0]

// OUT: [3]

// OUT: [-6.0] [3]

// OUT: [-8.0] [4.0]

Aufgabe 14) numerische Umwandlung

```
int k;
int drei = 3, vier = 4;
double pi = 3.14;
boolean t = true, f = false;
String anna = "Anna";
```

Ausdruck	Wert	Typ
`b = 0.5 < 1`	`true`	`boolean`
`drei + pi`		
`anna + "pi"`		
`anna + pi`		
`anna + 2 + 3`		
`2 + 3 + anna`		
`pi = 3`		
`k = (int) 0.5`		
`k = 0.5`		
`k = 5 < 1`		
`( k = 5 ) < 1`		
`f || !( f && t )`		
`t == true = false`		
`t == true == false`		
`t = true == false`		
`f != true`		
`f =! true`		
`!f == true`		
`!f = true`		
`!( f == true )`		
`!( f = true )`		

Aufgabe 15) Parameterübergabe und Rückgabewerte

```
class C {
  int c;

  C succ( ) {
    c += 1;
    return this;
  }

  int twice( ) {
    return c * 2;
  }

  void fill(
    for ( int
      a[ i ]
  }

  void fill(
    z = c;
  }
}
```

```
A[] a = { new A( ), new B( ), new C( ) };
```

(1)
```
for ( int i = 0; i < a.length; i++ ) {
  System.out.println( a[ i ].getString( ) );
}
```

(2)
```
for ( int i = 0; i < a.length; i++ ) {
  System.out.println( a[ i ].string );
}
for ( int i = 0; i < a.length; i++ ) {
  a[ i ].setString( "neu" );
}
```

(3)
```
for ( int i = 0; i < a.length; i++ ) {
  System.out.println( a[ i ].getString( ) );
}
```

(4)
```
for ( int i = 0; i < a.length; i++ ) {
  System.out.println( a[ i ].string );
}
```

```
C x = new C( );
System.out.println
System.out.println
System.out.println
System.out.println
System.out.println
System.out.println( x.succ( ).succ( ).twice( ) );
System.out.println( x.c );

x.c = 42;
int[] a = { 7 };
x.fill( a );
System.out.println( a[ 0 ] );

int z = 7;
x.fill( z );
System.out.println( z );
```

Aufgabe 16) Vererbung Methodenaufrufe

Was wird in den folgenden Programmstücken jeweils ausgegeben?

```
class A {
  String string = "oben";
  String getString( ) {
    return string;
  }
  void setString( String s ) {
    string = s;
  }
}

class B extends A {
  String string = "linksUnten";
  String getString( ) {
    return string;
  }
  void setString( String s ) {
    string = s;
    super.string = "auch " + s;
  }
}

class C extends A {
  String string = "rechtsUnten";
  String getString( ) {
    return string;
  }
  void setString( String s ) {
    string = s;
  }
}
```

Lösung:

// 1. oben linksUnten rechtsUnten

// 2. oben oben oben

// 3. neu neu neu

// 4. neu oben oben

Aufgabe 17) Statische Fehler

```
class Lemming {
```

```
(1) Lemming huey = new Lemming( "Huey" );
(2) System.out.println( huey.sayHello( ) );
(3) Lemming dewey = new Lemming( "Dewey" );
(4) dewey.bye( );
(5) System.out.println( huey.sayHello( ) );
(6) Lemming louie = new Lemming( "Louie" );
(7) System.out.println( louie.sayHello( ) );
```

```
    if ( alive )
      return "Hello, my name is " + name;
    else
      return "splash";
  }

  void bye( ) {
    alive = false;
  }
}
```

Was wird in den Zeilen 2, 5 bzw. 7 des folgenden Programmstücks ausgegeben?

Lösung:

// Hello, my name is Huey

// splash

// splash