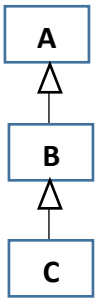
1. Aufgabe - Reihenfolge von Konstruktoraufrufen

Die folgende Klasse `TestKonstruktoren` dient zum Testen der Aufrufreihenfolge von Konstruktoren bei abgeleiteten Klassen.

```
// Datei: TestKonstruktoren
public class TestKonstruktoren {
   public static void main (String[] args) {
      System.out.println ("Exemplar von A wird angelegt");
      A aRef = new A();
      System.out.println();
      System.out.println ("Exemplar von B wird angelegt");
      B bRef = new B();
      System.out.println();
      System.out.println ("Exemplar von C wird angelegt");
      C cRef = new C();
      System.out.println();
   }
}
```



Betrachtete Klassenhierarchie

Schreiben Sie die 3 Klassen A, B und C, welche jeweils nur einen Konstruktor ohne Parameter enthalten. Im Konstruktor der Klasse A soll folgender Text ausgegeben werden:

```
System.out.println ("Klasse A - Konstruktor ohne Parameter");
```

Schreiben Sie entsprechende Konstruktoren für die Klassen B und C. Beachten Sie, dass B von A und C von B abgeleitet ist. Der Konstruktor der Klasse C soll den Default-Konstruktor der Basisklasse B durch `super()` explizit aufrufen. Beim Ausführen der Klasse `TestKonstruktoren` wird deutlich, dass es in diesem Fall keinen Unterschied macht, ob der Aufruf `super()` explizit durch den Programmierer eingefügt wird.

Schreiben Sie ein Testprogramm, das jeweils ein Objekt der Klassen A, B und C anlegt und untersuchen Sie die Ergebnisse.

2. Aufgabe

Nun wird es interessant. Gegeben sind die folgenden Java-Klassen:

```
class Maus {
	Maus() {System.out.println("Maus");}
}

class Katze {
	Katze() {System.out.println("Katze");}
}

class Ratte extends Maus {
	Ratte() {System.out.println("Ratte");}
}

class Fuchs extends Katze {
	Fuchs() {System.out.println("Fuchs");}
}

public class Hund extends Fuchs {
	Maus m = new Maus();
	Ratte r = new Ratte();
	Hund() {System.out.println("Hund");}

	public static void main(String[] args) {
		new Hund();
	}
}
```

Überlegen Sie zuerst was beim Start der Klasse Hund ausgegeben wird. Probieren Sie es dann aus.

3. Aufgabe

Ein produzierender Betrieb verwaltet seine hergestellten Produkte zurzeit mit folgenden drei Klassen:

```
public class Membranpumpe {
   private String name;
   private int tiefe;
   private float maximalerBetriebsdruck;
   private int hoehe;
   private String membranmaterial;
   private int gewicht;
   private int maximaleFoerdermenge;
   private int breite;
}

public class Kreiselpumpe {
   private int breite;
   private int hoehe;
   private int gewicht;
   private int anzahlSchaufeln;
   private int maximaleFoerdermenge;
   private int maximaleDrehzahl;
   private String name;
   private int tiefe;
   private float maximalerBetriebsdruck;
}

public class Auffangbecken {
   private int tiefe;
   private int volumen;
   private int breite;
   private int gewicht;
   private String name;
   private int hoehe;
}
```

a. Entwickeln Sie eine passende Vererbungshierarchie, welche die gemeinsamen Attribute in Basisklassen zusammenfasst.
b. Erweitern Sie alle Klassen der Vererbungshierarchie mit Konstruktoren, um eine einfache Initialisierung der Klassen zu ermöglichen.

4. Aufgabe

Warum kann der folgende Quellcode nicht übersetzt werden?

```
public class Vater {
   int iv;
   Vater(int ivv) {
      iv = ivv;
   }
   void hallo() {
      System.out.println("Hallo-Methode der Klasse Vater");
   }
}

class Tochter extends Vater {
   int it = 3;
   void hallo() {
      System.out.println("Hallo-Methode der Klasse Tochter");
   }
}
```

5. Aufgabe

Im folgenden Beispiel erlaubt der Compiler dem Kreis-Konstruktor keinen Zugriff auf die geerbten Instanzvariablen xpos und ypos eines neuen Kreis-Objekts.:

```
package f1;
public class Figur {
      double xpos, ypos;
}

package f2;
import f1.Figur;
public class Kreis extends Figur {
      double radius;
      Kreis(double x, double y, double rad) {
            xpos = x;
            ypos = y;
            radius = rad;
      }
}
```

Wie ist das Problem zu erklären und zu lösen?

6. Aufgabe

Wird in einer Oberklasse die Implementation einer Methode verbessert, profitieren auch alle abgeleiteten Klassen. Was muss geschehen, damit die Objekte einer abgeleiteten Klasse bei einer geerbten Methode die verbesserte Variante benutzen?

- Genügt es die Oberklasse neu zu übersetzen und (z.B. per Klassensuchpfad) dafür zu sorgen, dass die aktualisierte Basisklasse von der JRE geladen wird?
- Muss man sowohl die Basisklasse als auch die abgeleitete Klasse neu übersetzen?

Fragen zum Wiederholen/ Selbststudium/ Ausprobieren

Konstruktoraufrufe

1. Frage

Welche der untenstehenden Aussagen treffen nicht zu?

a) Der Aufruf super() muss im Konstruktor immer an erster Stelle stehen.
b) Existieren nur die impliziten Standardkonstruktoren, so wird automatisch im Standardkonstruktor der Subklasse der Konstruktor der Superklasse mit super() aufgerufen.
c) Es existiert immer ein Standardkonstruktor.
d) Mit this() wird der Standardkonstruktor der eigenen Klasse aufgerufen.
e) Mit super.laufen() wird die Methode laufen() der Superklasse aufgerufen.
f) Keine dieser Möglichkeiten.

2. Frage

Welche der untenstehenden Aussagen trifft zu?

a) Ein fehlender Standardkonstruktor kann nie zu Kompilierfehlern führen.
b) Der Aufruf super() muss im Konstruktor immer an letzter Stelle stehen.
c) Existieren nur die impliziten Standardkonstruktoren, so wird automatisch im Standardkonstruktor der Subklasse der Konstruktor der Superklasse mit this() aufgerufen.
d) Es existiert immer ein Standardkonstruktor.
e) Mit this() wird der Standardkonstruktor der Superklasse aufgerufen.
f) Keine dieser Möglichkeiten.

3. Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
      Vater(){
            System.out.println(“Vater“);
      }
}

class Tochter extends Vater{
      Tochter(){
            System.out.println(“Tochter“);
      }

      public static void main(String[ ] args){
            Tochter tochter = new Tochter();
      }
}
```

a) Tochter
b) Vater Tochter
c) Tochter Vater

d) Vater
e) Keine Ausgabe
f) Kompilierfehler

4. Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
      Vater(){
            System.out.println("Vater");
      }
}

class Tochter {
      Tochter(){
            System.out.println("Tochter");
      }

      public static void main(String[ ] args){
            Tochter tochter = new Tochter();
      }
}
```

a) Tochter
b) Vater Tochter
c) Tochter Vater
d) Vater
e) Keine Ausgabe
f) Kompilierfehler

5. Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
      Vater(){
            System.out.println("Vater");
      }
}

class Tochter extends Vater{
      Tochter(){
            System.out.println("Tochter");
      }
      public static void main(String[ ] args){
            Tochter tochter = new Tochter();
            super();
      }
}
```

a) Tochter
b) Vater Tochter
c) Tochter Vater
d) Vater
e) Keine Ausgabe

f) Kompilierfehler

6. Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
      Vater(){
            System.out.println(“Vater“);
      }
}

class Tochter extends Vater{
      Tochter(){
            super()
            System.out.println(“Tochter“);
      }

      public static void main(String[ ] args){
            Tochter tochter = new Tochter();
      }
}
```

a) Tochter
b) Vater Vater Tochter
c) Vater Tochter Vater
d) Vater Tochter
e) Keine Ausgabe
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

7. Frage

Welches Ergebnis erhält man, wenn man folgendes Programm kompiliert und startet?

```
class Vater{
      Vater(){
            System.out.println(“Vater“);
      }
}

class Tochter extends Vater{
      Tochter(){
            this(“Tochter“);
            System.out.println(“Tochter“);
      }

      Tochter(String wort){
            System.out.println(“Ich bin “ + wort);
      }

      public static void main(String[ ] args){
            Tochter tochter = new Tochter();
      }
}
```

a) Ich bin Tochter
b) Vater Vater Ich bin Tochter
c) Vater Ich bin Tochter Vater
d) Vater Ich bin Tochter Tochter
e) Keine Ausgabe
f) Kompilierfehler
g) Keine dieser Möglichkeiten.

Überschreiben von Methoden

1. Frage

Welche der untenstehenden Bemerkungen treffen zu?

a) Private Methoden können überschrieben werden.
b) Final Methoden können überschrieben werden.
c) Static Methoden können überschrieben werden.
d) Keine Methode einer Superklasse kann in der Subklasse überschrieben werden.
e) Die Begriffe Überschreiben und Überladen können synonym verwendet werden.
f) Ich muss beim Überladen von Methoden öffentlicher werden.
g) Ich muss Methoden aus der Superklasse in der Subklasse implementieren.
h) Keine dieser Möglichkeiten.

2. Frage

Welche der untenstehenden Bemerkungen treffen zu?

a) Eine Subklasse erbt Methoden aus der Superklasse.
b) Final Methoden dürfen nicht überschrieben werden.
c) Methoden aus der Superklasse müssen in der Subklasse implementiert werden.
d) Methoden einer Superklasse können in der Subklasse überschrieben werden.
e) Die neue Methode muss den gleichen Namen, die gleichen Parameter und den gleichen Rückgabetyp haben.
f) Ich darf beim Überladen von Methoden nicht öffentlicher werden.
g) Ich darf eine protected Methode mit einer public Methode überschreiben.
h) Keine dieser Möglichkeiten.

3. Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter {
      void f() { System.out.println(”M“); }
}

public class Tochter extends Mutter{}

public class Familie {
      public static void main(String[] args) {
            Mutter m = new Mutter();
            Tochter t = new Tochter();
            m.f();
            t.f();
      }
}
```

a) MM
b) M
c) Nichts
d) Es wird eine Exception geworfen.

.

4. Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter{
      void f() { System.out.println(”M“); }
}

public class Tochter {
}

public class Familie{
      public static void main(String[] args) {
            Mutter m = new Mutter();
            Tochter t = new Tochter();
            m.f();
            t.f();
      }
}
```

a) MM
b) M
c) Nichts
d) Es wird eine Exception geworfen.

5. Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
public class Mutter {
      void f() { System.out.println(”M“); }
}

public class Tochter extends Mutter {
      void f() { System.out.println(”T“); }
}

public class Familie{
      public static void main(String[] args) {
            Mutter m = new Mutter();
            Tochter t = new Tochter();
            m.f();
            t.f();
      }
}
```

a) MT
b) TM
c) M
d) T
e) Nichts
f) Es wird eine Exception geworfen.

6. Frage

Was wird bei untenstehenden Klassen ausgegeben?

```
class Mutter{
      void f() { System.out.println(”M“); }
}

public class Tochter {
      void f() { System.out.println(”T“); }
}

public class Familie{
      public static void main(String[] args) {
            Mutter m = new Mutter();
            Tochter t = new Tochter();
            m.f();
            t.f();
      }
}
```

a) MT
b) TM
c) M
d) T
e) Nichts
f) Es wird eine Exception geworfen.

7. Frage

Welche Methoden, die die Methode f() aus der Klasse Mutter überschreiben, dürfen in der Klasse Tochter stehen, ohne dass es zu einem Kompilierfehler kommt?

```
public class Mutter {
      void f() { System.out.println(“M”); }
}

public class Tochter extends Mutter {
      Welche Methode darf hier stehen?
}
```

a) private void f( ){ }
b) public void f( ){ }
c) protected void f( ){ }
d) int f( ) {}
e) Keine dieser Möglichkeiten.

8. Frage

Welche Methoden, die die Methode f() aus der Klasse Mutter überschreiben, dürfen in der Klasse Tochter stehen, ohne dass es zu einem Kompilierfehler kommt?

```
public class Mutter{
      public void f() { System.out.println("M"); }
}

public class Tochter extends Mutter {
      Welche Methode darf hier stehen?
}
```

a) private void f( ){ }
b) public void f( ){ }
c) protected void f( ){ }
d) int f( ) {}
e) Keine dieser Möglichkeiten.

9. Frage

Welche Methoden, die die Methode f() aus der Klasse Mutter überschreiben, dürfen in der Klasse Tochter stehen, ohne dass es zu einem Kompilierfehler kommt?

```
public class Mutter {
      private void f() { System.out.println("M"); }
}

public class Tochter extends Mutter {
      Welche Methode darf hier stehen?
}
```

a) final void f( ){ }
b) static void f( ){ }
c) void f( ){ }
d) string f( ) {}
e) Keine dieser Möglichkeiten.