JAVA F01: Gemischte komplexere Fragen zu Java

panitz

Zusammenfassung

Die Karten dieses Kurses sind vom Alexander Gepperth an der FH Fulda übernommen und für Subato erweitert worden.

Frage: Schleifen

Geben Sie an, wie oft durch den folgenden Code die Ausgabe von »Y« erfolgt.

```
public class  A21{
  public static void main (String[] args) {
    for (int i= 22; i > 10 ; i--) {
      System.out.println( (i > 16) ? "X" : "Y") ;
    }
  }
}
```

Korrekte Antworten

- 6

Erläuterung

In der Schleife wird die Laufvariable `i` von 22 bis 11 herunter gezählt. Durch den Bedingungsoperator wird aber erst ab dem Wert 16 ein »Y« auf der Kommandozeile ausgedruckt.

Frage: Abbruchbedingung

Was ist die Ausgabe des folgenden Programms?

```
public class A22{
  public static void main (String[] args) {
    int i = 5;
    while (i < 23) {
      if (i > 14) {
        break ;
      }
      i += 2 ;
    }
    System.out.println(i) ;
  }
}
```

Korrekte Antworten

- 15

Erläuterung

Die Schleife wird auf jeden Fall nicht regulär beendet, weil die Schleifenbedingung nicht mehr wahr ist, sondern, weil die if-Bedingung wahr ist und die Schleife durch die `break`-Anweisung verlassen wird.

Frage: Abbruchbedingung

Wie oft schreibt das Code-Snippet "Hallo"?

```
int k = 0;
while(k < 23) {
  k++ ;
  if (k > 11) {
    continue ;
  }
  System.out.println("Hallo") ;
}
```

Korrekte Antworten

- 11

Erläuterung

Wenn die if-Bedingung wahr ist, wird der Schleifenrumpf für den aktuellen Durchlauf durch die `continue`-Anweisung verlassen. Der Code nach der if-Bedigung wird in diesem Durchlauf dann nicht mehr ausgeführt.

Frage: Schleifen

Wie oft schreibt das Code-Snippet "Hallo"?

```
int z = 0;
do {
  System.out.println("Hallo") ;
  z++ ;
} while (z < 7);
```

Korrekte Antworten

- 7

Erläuterung

Frage: Referenzen

Was ist die Ausgabe des folgenden Code-Snippets?

```
ArrayList<Integer> list1 = new ArrayList<Integer>();
ArrayList<Integer> list2 = list1;
list1.add(3);
list2.add(5);
System.out.println(list1.size());
```

Korrekte Antworten

- 2

Erläuterung

Es gibt nur ein Listenobjekt, das über zwei Variablen angesprochen wird. In diese Liste werden zwei Elemente eingefügt.

Frage: Referenzdatentypen

Welche der genannten Datentypen sind keine Referenzdatentypen?

(Mehrere Antwortmöglichkeiten).

- Integer
- Object
- int
- char
- String
- int []

Korrekte Antworten

- int
- char

Erläuterung

In Java gibt es 8 primitiven Datentypen die nicht als Refrenzen übergeben werden.

Frage: Referenzdatentypen

Welche der genannten Datentypen sind Referenzdatentypen?

(Mehrere Antwortmöglichkeiten).

- `ArrayList<Integer>`
- `LinkedList<String>`
- `float`
- `char`
- `String[]`
- `int[]`
- `float[]`

Korrekte Antworten

- `ArrayList<Integer>`
- `LinkedList<String>`
- `String[]`
- `int[]`
- `float[]`

Erläuterung

In Java sind alle Datentypen bis auf die 8 primitiven Datentypen Referenzdatentypen. Das gilt insbesondere auch für Arrays.

Frage: Statische Attribute

Was ist die Ausgabe des folgenden Programms?

```
class KlausurKlasse41 {
  public static int staticAttr;
  public KlausurKlasse41(int i) {
    this.staticAttr = i;
  }
  public static void main(String[] args) {
    KlausurKlasse41 k1 = new KlausurKlasse41 (1) ;
    KlausurKlasse41 k2 = new KlausurKlasse41 (5) ;
    KlausurKlasse41 k3 = new KlausurKlasse41 (3) ;
    KlausurKlasse41 k4 = new KlausurKlasse41 (0) ;
    System.out.println(k1.staticAttr) ;
  }
}
```

Korrekte Antworten

- 0

Erläuterung

Statische Felder existieren für eine Klasse nur einmal. Alle Objekte der Klasse teilen sich dabei einen Speicherplatz und haben dort Zugriff auf dieselben Daten.

Frage: Sichtbarkeit von Methoden

Geben Sie an, ob der unten stehende Code erfolgreich compiliert werden kann oder nicht. (Die Methode size() bestimmt die Anzahl der in einer ArrayList gespeicherten Elemente.)

```
import java.util.* ;
public class Klausur42 {
  public static void main(String[] args) {
    Object referenz = new ArrayList<Integer>() ;
    System.out.println(referenz.size()) ;
  }
}
```

(Eine Antwortmöglichkeit.)

- Der Code kompiliert ohne Fehler aber verursacht eine Exception zur Laufzeit.
- Der Code kompiliert ohne Fehler und wird fehlerfrei ausgeführt.
- Der Code kompiliert nicht fehlerfrei, weil ArrayList niemals den Typparameter Integer bekommen darf.
- Der Code kompiliert nicht fehlerfrei, weil Object die Methode size nicht besitzt.
- Der Code kompiliert nicht fehlerfrei, weil *referenz* vom Typ Object ist aber eine Instanz vom Typ ArrayList zugewiesen bekommt.
- Der Code kompiliert nicht fehlerfrei weil ArrayList mit einem Größenparameter instanziert werden muss

Korrekte Antworten

- Der Code kompiliert nicht fehlerfrei, weil Object die Methode size nicht besitzt.

Erläuterung

Die Variable `referenz` ist vom Typ `Object` deklariert. Dort ist zwar ein Objekt der Klasse `Arraylist` gespeichert, der statische Typcheck weiß aber nur, dass dort statisch irgend ein Objekt gespeichert wurde. `size` ist keine Methode der Klasse `Object`.

Frage: Überschreiben von Methoden

Was ist die Ausgabe des folgenden Programms?

```
class Frucht {
  int getWeight() {  return 0 ;}
}
class Melone extends Frucht {
  @Override int getWeight() { return 1500;}
}
class Weintraube extends Frucht {
  @Override int getWeight() { return 10;}
}
class Apfel extends Frucht {
  @Override  int getWeight() { return 500 ;}
  public static void main(String[] args) {
    Frucht frucht1 = new Apfel() ;
    Frucht frucht2 = new Weintraube() ;
    frucht1 = new Melone() ;
    frucht1 = frucht2 ;
    System.out.println(frucht1.getWeight()) ;
  }
}
```

Korrekte Antworten

- 10

Erläuterung

Im Endeffekt ist die Referenz im Feld `frucht1` eine Referenz auf ein Objekt der Klasse `Weintraube`. Daher wird durch die späte Bindung (late binding) die Methode `getWeight` der Klasse `Weintraube` ausgeführt.

Frage: Exception Handling

Was ist die letzte Ausgabe des folgenden Programms?

```
import java.io.* ;
class KlausurKlasse51 {
  public static void methode1() throws IOException {
    throw new IOException() ;
  }
  public static void main(String [] args) {
    try {
      methode1() ;
      System.out.println("1") ;
    } catch (RuntimeException e) {
      System.out.println("0") ;
      return ;
    } catch (IOException e) {
      System.out.println("3") ;
      return ;
    }
    System.out.println("4") ;
  }
}
```

Korrekte Antworten

- 3

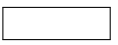
Erläuterung

Die Methode `methode1` wirft auf jedem Fall eine Ausnahme der Klasse `IOxception`. Diese wird im entsprechendem `catch` gefangen. Dort wird nach der Ausgabe die Methode `main` mit einer `return`-Anweisung beendet.

Frage: Iteratoren

Was ist die Ausgabe des folgenden Programms?

```
LinkedList<Integer> list = new LinkedList<Integer>();
list.add(5); list.add(6); list.add (7); list.add(0);
Iterator<Integer> it = list.iterator();
int tmp = it.next() ;
System.out.println(it.next() + "" + it.next()) ;
```



Korrekte Antworten

- 67

Erläuterung

Die Liste enthält die Elemente 5, 6, 7, 0. Jeder Iterator der Liste iteriert über diese Elemente. Ein Iterator gibt beim Aufruf der Methode `next` nihct nur das nächste zu iterierende Element zurück, sondern schaltet den Iterator ein Element weiter.

Frage: Iteratoren

Betrachten Sie das unten stehende Code-Fragment. Geben Sie an, welche Aussage zum Code-Fragment korrekt ist.

```
LinkedList<Double> list = new LinkedList<Double>() ;
list.add(5.0); list.add(6.0); list.add (7.0);
Iterator<Double> it = list.iterator() ;
it.next() ; it.next() ; it.next();
System.out.println(it.next());
```

(Mehrere Antwortmöglichkeiten).

- Der Code kann fehlerfrei kompiliert werden
- Der Code kann fehlerfrei ausgeführt werden
- Der Code kann nicht fehlerfrei ausgeführt werden
- Beim Ausführen wird eine NoSuchElementException verursacht
- Beim Ausführen wird eine RuntimeException verursacht
- Beim Ausführen wird eine ArrayIndexOutOfBoundsException verursacht

Korrekte Antworten

- Der Code kann fehlerfrei kompiliert werden
- Der Code kann nicht fehlerfrei ausgeführt werden
- Beim Ausführen wird eine NoSuchElementException verursacht
- Beim Ausführen wird eine RuntimeException verursacht

Erläuterung

Der Code generiert eine Exception, weil der Iterator das Ende der LinkedList erreicht hat. Abfrage von hasNext() vor jedem Aufruf von next() könnte dieses verhindern. Die Klasse `NoSuchElementException` erbt von der Klasse `RuntimeException`.

Frage: Code Style

Wie müsste folgendes Programmfragment im Sinne des *Google Java Style* korrekt aussehen? Nur eine Antwort ist richtig, stehen zur besseren Lesbarkeit für Leerzeichen.

(Eine Antwortmöglichkeit.)

- `while (i<150 && k<100)`
- `while(i < 150 && k < 10)`
- `while(i < 150 && k < 10)`
- `while (i < 150 && k < 10)`
- `while (i < 150 && k<10)`
- `while (i<150&&k<10)`

Korrekte Antworten

- `while (i < 150 && k < 10)`

Erläuterung

Insbesondere Operatoren sollen durch Leerzeichen abgesetzt werden. Auch zwischen den Schlüsselwörtern und Klammersymbolen sollen Leerzeichen stehen.

Frage: Dokumentation

Welche Elemente müssten für die folgenden Methode in einem JavaDoc-Kommentar (nach den Regeln des "Google Java Style") angegeben werden:

```
public double berechneWurzel(double x)
  throws IllegalArgumentException {
  if (x < 0) {
    throw new IllegalArgumentException ("Argument muss > 0 sein!") ;
  }
  return Math.sqrt(x) ;
}
```

(Mehrere Antwortmöglichkeiten).

- Eine kurze Erklärung, was diese Methode macht, gefolgt von einem Punkt.
- `@param x double Eine Zahl`
- `@return Die Wurzel`
- `@throws IllegalArgumentException Falls x < 0`
- `@throw IllegalArgumentException Falls x < 0`
- `@param x float Eine Zahl`
- `@returns Das Ergebnis`

Korrekte Antworten

- Eine kurze Erklärung, was diese Methode macht, gefolgt von einem Punkt.
- `@param x double Eine Zahl`
- `@return Die Wurzel`
- `@throws IllegalArgumentException Falls x < 0`

Erläuterung

Dieses ist eine Beispieldokumentation:

```
/** Diese Methode berechnet die Quadratwurzel.
  * @param x double Eine Zahl
  * @return double Die Wurzel
  * @throws IllegalArgumentException Falls x < 0
  */
```

Frage: Dokumentation

Geben Sie den JavaDoc-Kommentar (nach den Regeln des "Google Java Style") an, der vor folgender Methode stehen müsste:

```
Integer berechneMaximum(Integer i, Integer j)
    throws IllegalArgumentException {
  if ((i < 0) || (j < 0)) {
    throw new IllegalArgumentException("Zahlen sollen > 0 sein!");
  }
  if (i >= j) {
    return i;
  } else  return j;
}
```

(Mehrere Antwortmöglichkeiten).

- Diese Methode berechnet das Maximum zweier Zahlen.
- `@param i Integer Eine Zahl`
- `@param j Integer Noch eine Zahl`
- `@param k Integer Noch eine Zahl`
- `@return  die kleinere der beiden Zahlen`
- `@returns  die kleinere der beiden Zahlen`
- `@throws IllegalArgumentException Falls i oder j < 0`
- `@throw IllegalArgumentException Falls i oder j < 0`
- Ein ausführliches Tuturial, wie diese Methode verwendet werden sollte

Korrekte Antworten

- Diese Methode berechnet das Maximum zweier Zahlen.
- `@param i Integer Eine Zahl`
- `@param j Integer Noch eine Zahl`
- `@return  die kleinere der beiden Zahlen`
- `@throws IllegalArgumentException Falls i oder j < 0`

Erläuterung

Beispieldokumentation:

```
/** Diese Methode berechnet das Maximum zweier Zahlen.
  * @param i Integer Eine Zahl
  * @param j Integer Noch eine Zahl
  * @return Integer die groessere der beiden Zahlen
  * @throws IllegalArgumentException Falls i oder j < 0
  */
```

Frage: Gelöscht

Betrachten Sie das unten stehende Code-Fragment und erläutern Sie mit Begründung, ob und welches Problem nach einem darauf folgenden Aufruf von System.out.println(it.next()) auftritt. Erläutern Sie weiter, welche Maßnahme dieses Problem verhindern könnte!

```
var list = new LinkedList<Double>() ;
list.add(5.0) ; list.add(6.0); list.add (7.0);
Iterator<Double> it = list.iterator() ;
it.next() ; it.next() ; it.next() ;
```

(Eine Antwortmöglichkeit.)

- Es tritt kein Problem auf
- Compiliert nicht aufgrund von Syntaxfehlern
- Es tritt eine NoSuchElementException auf, welche durch vorherige Abfrage von hasNext verhindert werden kann
- Es tritt eine IllegalArgumentException auf. Um sie zu verhindern müsste ein neuer Iterator erzeugt werden
- Es tritt eine NullPointerException auf. Um dies zu verhindern muss der Iterator im Inneren einer for-Schleife abgefragt werden.

Korrekte Antworten

- Es tritt eine NoSuchElementException auf, welche durch vorherige Abfrage von hasNext verhindert werden kann

Erläuterung

Frage: Arrays

Was ist die Ausgabe des folgenden Programms?

```
public class A42 {
  public static void macheWas(int[] arr) {
    arr[0] = 1 ;
  }

  public static void main (String[] args) {
    int [] x = {0,0,0,0} ;
    int [] y = {2,2,2,2} ;
    int [] z = x ;
    macheWas(z) ;
    System.out.println(x[0]) ;
  }
}
```

Korrekte Antworten

- 1

Erläuterung

Arrays werden immer als Referenzen übetrgeben.

Frage: Statische Attribute

Was ist die Ausgabe des folgenden Programms?

```
class A51{
  public static int staticAttr ;
  public void setAttr(int i) {
    A51.staticAttr = i;
  }
  public static void main(String[] args) {
    A51 k1 = new A51 () ;
    k1.setAttr(5) ;
    A51 k2 = new A51 () ;
    k2.setAttr(2) ;
    System.out.println(A51.staticAttr) ;
  }
}
```

Korrekte Antworten

- 2

Erläuterung

Frage: Typ-Zuweisung

Geben Sie an ob das folgende Programm compiliert werden kann!

```
import java.util.* ;
public class Klausur42 {
  public static void main(String[] args) {
    ArrayList <Integer> referenz = new Object() ;
    System.out.println(referenz.toString()) ;
  }
}
```

(Eine Antwortmöglichkeit.)

- Compiliert nicht da eine gescheifte Klammer am Ende fehlt
- Compiliert fehlerfrei
- Compiliert nicht da eine Referenz vom Typ ArrayList<Integer> nicht auf eine Instanz vom Typ Object zeigen darf
- Compiliert nicht da eine Referenz vom Typ Object nicht auf eine Instanz vom Typ ArrayList<Integer> zeigen darf
- Compiliert nicht da die Klasse java.util.ArrayList explizit importiert werden muss
- Compiliert nicht da Object keine Methode toString() besitzt

Korrekte Antworten

- Compiliert nicht da eine Referenz vom Typ ArrayList<Integer> nicht auf eine Instanz vom Typ Object zeigen darf

Erläuterung

Frage: Vererbung

Was ist die Ausgabe des folgenden Programms?

```
class Frucht {
  int getWeight() { return 0 ;}
}
class Melone extends Frucht {
  @Override  int getWeight() {return 1500;}
}
class Weintraube extends Frucht {
  @Override  int getWeight() {return 10;}
}
class Apfel extends Frucht {
  @Override  int getWeight() {return 500;}
  public static void main(String[] args) {
    Frucht frucht1 = new Apfel() ;
    Frucht frucht2 = new Weintraube() ;
    Frucht frucht3 = frucht1 ;
    frucht2 = new Melone() ;
    System.out.println(frucht3.getWeight()) ;
  }
}
```

Korrekte Antworten

- 500

Erläuterung

Frage: Exceptions

Was ist die Ausgabe des folgenden Programms?

```
import java.io.*;
public class A61 {
  public static int exceptional(int i) throws IllegalArgumentException {
    if (i<0) {
      throw new IllegalArgumentException() ;
    } else {
      return i+1 ;
    }
  }
  public static void main(String [] args) {
    try {
      System.out.println("1");
      int j = exceptional(-2);
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("2");
    } catch (IllegalArgumentException e) {
      System.out.println("3");
    }
    System.out.println("5");
  }}
```

Korrekte Antworten

- 135

Erläuterung

Frage: Iteratoren

Was ist die Ausgabe des folgenden Programms?

```
import java.util.* ;
public class A61 {
  public static void main(String[] args) {
    LinkedList<Integer> list = new LinkedList<Integer>() ;
    list.add(5) ; list.add(8); list.add (2);
    Iterator<Integer> it = list.iterator() ;
    it.next() ; it.next() ;
    for ( ; it.hasNext() == true; ) {
      System.out.println(it.next()) ;
    }
  }
}
```

Korrekte Antworten

- 2

Erläuterung

Frage: Iteratoren

Betrachten Sie das unten stehende Code-Fragment und erläutern Sie mit Begründung, wie viele Aufrufe von it.next() nach der Ausführung des letzten Kommandos noch ohne Probleme durchgeführt werden können.

```
LinkedList<Double> list = new LinkedList<Double>();
list.add(5.0) ; list.add(6.0); list.add (7.0); list.add(0.0);
Iterator<Double> it = list.iterator();
it.next();
```

(Eine Antwortmöglichkeit.)

- 0
- 1
- 2
- 3
- 4
- 5

Korrekte Antworten

- 3

Erläuterung

Noch 3 Aufrufe möglich, danach Exception, weil der Iterator das Ende der Liste erreicht hat.

Frage: Ternärer Operator

Ist die folgende Ersetzung der if-Bedingung durch einen ternären Operator korrekt umgesetzt worden?

```
int methode1(int z) {
  if (z != 3) {
    return 0;
  } else {
    return 1 ;
  }
}
```

```
int methode1(int z) {
  return (z != 3) ? 0 : 1 ;
}
```

(Eine Antwortmöglichkeit.)

- Ja
- Nein, da die Rückgabewerte 0 und 1 vertauscht sind.
- Nein, da immer 0 zurückgegeben wird.
- Nein, da immer 1 zurückgegeben wird.
- Nein, da beim Ternären Operator keine Vergleiche mit != erlaubt sind.

Korrekte Antworten

- Ja

Erläuterung

Frage: Ternärer Operator

Ist die folgende Ersetzung der if-Bedingung durch einen ternären Operator korrekt umgesetzt worden?

```
String methode2(String x) {
  if (x.equals("hallo")) {
    return "hallo!";
  } else {
    return "tschuess!" ;
  }
}
```

```
String methode2(String x) {
  return (x.equals("hallo")) ? "hallo!" : "tschuess!" ;
}
```

(Eine Antwortmöglichkeit.)

- Ja
- Nein, da immer "tschuess" zurückgegeben wird
- Nein, da immer "hallo" zurückgegeben wird
- Nein, da beim Ternären Operator keine Referenzdatentypen erlaubt sind
- Nein, da beim Ternären Operator keine Vergleiche mit equals erlaubt sind

Korrekte Antworten

- Ja

Erläuterung

Frage: Ternärer Operator

Ersetzen Sie die if-Anweisung durch den ternären Operator!

```
float kompliziertеBedingung(float i) {
  if(i < 0.0f) {
    return -1f ;
  } else if (i < 3f) {
    return i ;
  } else return 1f ;
}
```

```
float kompliziertеBedingung(float i) {
  return (i < 0f)? -1f : ((i < 3f) ? 1f : i) ;
}
```

(Eine Antwortmöglichkeit.)

- Ja
- Nein, da die Rückgabe von i und 1f vertauscht ist
- Nein, da die Rückgabe von i und 0f vertauscht ist
- Nein, da 0f und 0.0f nicht den gleichen Wert repräsentieren
- Nein, da immer 0f zurückgegeben wird
- Nein, da beim Ternären Operator keine Verschachtelung möglich ist

Korrekte Antworten

- Nein, da die Rückgabe von i und 1f vertauscht ist

Erläuterung

Frage: Überschreiben von Methoden

Was ist die Ausgabe des folgenden Programms?

```
class ParentClass {
  void printInfo(){System.out.println("Info Oberklasse");}
}
class ChildClass extends ParentClass  {
  @Override void printInfo() {
    System.out.print("Info Unterklasse ") ;
  }
  public static void main(String [] args) {
    ParentClass parent = new ChildClass() ;
    parent.printInfo() ;
    ChildClass child = new ChildClass() ;
    child.printInfo() ;
  }
}
```

(Eine Antwortmöglichkeit.)

- Info Oberklasse Info Oberklasse
- Info Oberklasse Info Unterklasse
- Info Unterklasse Info Unterklasse
- Info Unterklasse Info Oberklasse

Korrekte Antworten

- Info Unterklasse Info Unterklasse

Erläuterung