

Frage 70

Mittwoch, 18. Dezember 2024 14:47

```
class App{
    /* comment text 1 */
    // comment Text 2 //
    // comment text 3
    and comment text 4 //
    /* comment text 5
        and comment text 6 */
}
```

At which line does a compilation error occur?

- line 3
- line 7
- line 2
- line 5 (text 4)

Wieso?:

Der Text in Zeile 5 beginnt weder mit // noch ist Zeile 5 Teil eines Kommentarblocks /* */

Frage 71

Dienstag, 3. Dezember 2024 19:24

Given:

```
public static void main(String[] args){  
    boolean value1 = 10 + 5 >= 2 + 13;  
    int    value2 = 0;  
    if (value1 == true){  
        value2 = 5 * 3 + 10 / 2;  
    }else{  
        value2 = 5 / 3 + 10 * 2;  
    }  
    System.out.println(value2);  
}
```

What is the result?

- 32
- 21
- 20
- A compilation error occurs.

Wieso?:

$10+5=15$, $2+13 = 15$, $15 \geq 15 \rightarrow \text{value1} = \text{true}$
 $(5*3)+(10/2)$ (Punkt vor Strich)
 $15+5 = 20 \rightarrow \text{value2} = 20$

Frage 72

Dienstag, 3. Dezember 2024 19:24

Given the classes:

Apple.java:

```
package fruits;  
class Apple{  
    public void getApple(){}
}
```

Salad.java:

```
package food;  
//line n1  
  
class Salad{  
    Apple apple = new Apple(); //line n2  
    public void prepareSalad{  
        apple.getApple();
    }
}
```

Which two modifications, independently, enable the Salad.java file to compile?

- A Replace line n1 with import fruits.Apple;
- B Replace line n2 with fruits.Apple apple = new Apple ();
- C Replace line n1 with import fruits.Apple.getApple();
- D Replace line n1 with import fruits;
- E Replace line n2 with fruits.Apple apple = new fruits.Apple ();

Wieso?:

A und E, weil es entweder Sinn macht die Klasse Apple zu importieren oder einen Inline-Import durchzuführen.

Dies macht Sinn, wenn nur einmal benötigt oder zwei gleichnamige Klassen aus unterschiedlichen Packages gebraucht werden.

Bei B fehlt das package fruits(.) zwischen new und Apple

Bei C kann man keine Methode importieren, außerdem ist sie nicht statisch.

Bei D reicht der Import des Packages nicht.

Frage 73

Dienstag, 3. Dezember 2024 19:26

Given:

```
String digits = "0123456789";
System.out.println(
    digits.substring(
        digits.indexOf("4"), digits.indexOf("8")).concat("89")
);
```

What is the result?

- A 3456789
- B 4567889
- C 456789
- D 45678

C, weil beim Substring das Intervall des Indizes von 4 und das von 8 angegeben wird,
wobei aber der zweite Bound exklusiv ist und die 8 sich somit nicht doppelt nach Anfügen von 89

Frage 74

Dienstag, 3. Dezember 2024 19:26

Given:

```
int[] num = new int[2];
num[0] = 10;
num[1] = 15;
```

```
List<Integer> lst = new ArrayList<>(2);
lst.add(10);
lst.add(15);
```

```
num[1] = 20;
lst.add(20);
```

```
for (int x : num) {
    System.out.print(x + " ");
}
System.out.println();
```

```
for (int y : lst) {
    System.out.print(y + " ");
}
```

What is the result?

- A. A compilation error occurs.
- B. A runtime exception is thrown.
- C.
10 20
10 20
- D.
10 20
10 15 20

D, weil die 15 im Array num mit 20 überschrieben wird und bei der ArrayList der Wert 20 hingegen angefügt wird.

Frage 75

Dienstag, 3. Dezember 2024 19:27

Given:

```
public class App {  
    public static void main(String[] args) {  
        System.out.println("Hello Java!");  
    }  
}
```

Which statement is true about the main method?

- A. It cannot be invoked by its name.
- B. Its parameter can be of type Integer [].
- C. It cannot be defined in a non-public class.
- D. It can be a non-static method.

A, weil die main der Einstiegspunkt ist und ein Aufruf von egal wo, immer rekursiv wäre, da man das Programm nicht ohne main ausführen kann.

B und D widersprechen der Pflichtsignatur der PSVM
C nicht, weil die JVM dann die main nicht finden kann.

Frage 76

Dienstag, 3. Dezember 2024 19:27

Given:

```
public class App {  
    public static void main(String[] args) {  
        String name = "Rita";  
        int age = 14;  
        // line n1  
    }  
}
```

Which code fragment, when inserted at line n1, enables it to print Rita is 14 years old?

Frage 76Antwort

- A. System.out.printf("%s is %d years old", name, age);
- B. System.out.printf("%s is %n years old", name, age);
- C. System.out.println("%s is %n years old" name, age);

(Arthur, Alex, Dome)

Lösung: A

Begründung:

Es geht nur A, weil bei B ist age nicht im template als %d
C geht nicht, weil println keine Formatierungstemplates unterstützt.

Frage 77

Dienstag, 3. Dezember 2024 19:27

Given:

```
public class TestFinal {  
    final int l = 5;  
    static void modify(TestFinal test) {  
        test.l = 99;  
    }  
    public static void main(String[] args) {  
        final TestFinal tf = new TestFinal();  
        modify(tf);  
        System.out.println(tf.l);  
    }  
}
```

What is the result?

- A. 99
- B. 5
- C. A compilation error occurs in the modify method.
- D. A compilation error occurs in the main method.

Der Compiler erkennt schon dort, dass illegalerweise versucht wird eine Konstante zu überschreiben.

Frage 78

Dienstag, 3. Dezember 2024 19:28

Given:

```
int count = 0;
while (count <= 10) {
    System.out.print(count + " ");
    /* line n1 */
}
```

Which statement, when inserted at line n1, enables the code to print 0 2 4 6 8 10?

- A. count += 2;
- B. count **+= 2**;
- C. count = count++;
- D. count = (count++) +1;

A ist keine gültige Kurzschreibweise.

C und D nicht, weil eine In- und Dekrementierung nicht über Zuweisung erfolgt, sondern alleinstehend, also i++; oder i--;

Außerdem werden die Ergebnisse der In- und Dekrementierung so nicht verwertet.

Selbst wenn doch, so wäre C auch nur C+=1

Frage 79

Dienstag, 3. Dezember 2024 19:28

Identify three advantages of object-oriented programming.

- A. separation of state and behavior
- B. information sharing
- C. information hiding**
- D. modularity**
- E. code reuse**

Zustände und Verhalten von Objekten werden in Klassen vereint.

Datenkapselung wirkt dem Offenliegen und (ungewolltem) Teilen von Informationen entgegen.

Modularität ermöglicht die Wiederverwendung von Code.

Frage 80

Dienstag, 3. Dezember 2024 19:28

Given:

```
List<String> names = new ArrayList<>();  
names.add("Joel");  
names.add("Paul");  
names.remove(0);  
names.remove(0);  
System.out.println(names.isEmpty());  
names.add("Joel");  
names.add("Paul");  
names.clear();  
System.out.println(names.isEmpty());
```

What is the result?

A.

true

true

B.

false

true

C.

A runtime exception is thrown

D.

false

false

Vor beiden Ausgaben werden die Listen erst befüllt und dann geleert.
Das clear erspart einem das manuelle Entfernen aller einzelnen Elemente.

Frage 81

Dienstag, 3. Dezember 2024 19:29

Given:

```
int a = 3;  
a = ++a + a++;  
a = --a - a--;  
System.out.println(a);
```

What is the output?

A. A compilation error occurs.

B. 8

C. 0

D. 3

Egal, wie man a initialisiert, es wird am Ende 0.

Die Pre-In-/Dekrementierung wird jeweils zuvor ausgeführt, also wird a zuerst zu 4 (von) und dann zu 7 (von 8).

Dabei findet die Post-In-/Dekrementierung jeweils keine Anwendung (wird quasi ignoriert).

Die Pre-In-/Dekrementierung wirkt sich sofort aus, nicht nur auf das erste a der Zuweisung, sondern auch direkt das zweite hinter dem Operator.

Hier rechnet es sich also wie folgt:

$$a = (+1+3) 4 + 4 = 8 \{8+1\}$$

$$a = (-1+8) 7 - 7 = 0 \{7-1\}$$

<https://g.co/gemini/share/804d4210d687>

Die Änderungen ++a und --a finden sofort statt und auch schon Anwendung nach dem Operator in selbiger Anweisung.

Wohingegen a++/a-- erst in der darauffolgenden Verwendung verändert wird.

Frage 82

Dienstag, 3. Dezember 2024 19:29

```
int num[] = new int[3];
num[1] = 10;
num[2] = 15;
List<Integer> lst = new ArrayList<>(3);
lst.add(10);
lst.add(15);
System.out.println(num);
System.out.println(lst);
```

What is the result?

A.

10, 15
[10, 15]

B.

a memory address1 in hexadecimal number format
a memory address2 in hexadecimal number format

C.

0, 10, 15
[10, 15, null]

D.

a memory address in hexadecimal number format
[10, 15]

Ein Array kann ohne seine Wrapperklasse nicht einfach als String ausgegeben werden.
Aufgrund dessen erhält man ersatzweise aus der impliziten `toString`-Methode aus der Objektklasse die Speicheradresse.

Man kann aber die `asList`, oder `toString` Methode durch den Wrapper aufrufen.

Als Gegenbeispiel stellt die `ArrayList` durch das Listen-Interface bereits den impliziten Listenstring zur Verfügung.

Frage 83

Dienstag, 3. Dezember 2024 19:29

Given:

```
Random r1 = new Random(10);
Random r2 = new Random(10); // line n1
if (r1.nextInt() == r2.nextInt()) {
    System.out.println("Jack");
} else {
    System.out.println("Queen");
}
```

What is the result?

- A. Jack
- B. Queen
- C. A compilation error occurs at line n1.
- D. The program prints either Jack or Queen.

Beide Randomizer erhalten stets den gleichen Wert, weil sie denselben Seed erhalten haben. Sie verweisen also immer auf dieselben Werte. Hier tritt also nie der else-Zweig ein.

Frage 84

Dienstag, 3. Dezember 2024 19:29

Which package would you import to use the Random class?

- A. java.io
- B. java.util(Richtig)**
- C. java.math
- D. java.lang

Es wurde zwar schon gesagt, aber nur dort ist die Klasse enthalten.

Das random in Math ist eine Methode und Math gehört zu java.lang.

Io hat mit Ein- und Ausgaben für Daten und Dateien zu tun.

Frage 85

Dienstag, 3. Dezember 2024 19:29

Which two components can class declarations include?

- A. A list of instance methods
- B. The main method
- C. The public modifier
- D. Interfaces implemented by the class

Beschreibe in eigenen Worten was gemeint ist:

Bei der Klassendeklaration ist von ihrer Signatur die Rede. Darin stehen der Modifier und Interface-Implementierungen.

Variablen und Methoden hingegen sind im Body der Klasse und können eher als "Initialisierung" verstanden werden.

Frage 86

Dienstag, 3. Dezember 2024 19:30

Given:

```
String test = "a"
for (; test.compareTo("aaa") == 0; test = test + "a") {
    System.out.print(test.length() + " ");
}
System.out.print(test);
```

What is the output?

- A. a
- B. 1 2 3 aaaa
- C. 1 2 aaa
- D. Compilation fails

Der Ausschnitt suggeriert, dass die Schleife laufen soll, bis aus a erst aa und dann aaa geworden ist.

Nun ist es aber so, dass die Weiterlaufbedingung direkt zu Beginn schon nicht erfüllt ist.

Wird nämlich beim compareTo an keinem Index ein alphabetischer (lexikografischer) Unterschied entdeckt, wird als zweites die Länge der Strings verglichen und der Längenunterschied stattdessen ausgegeben.

Findet sich vor Ende der Strings ein lexikografischer Unterschied, spielt eventueller Längenunterschied keine Rolle mehr.

Die Schleife wird hier also direkt übersprungen und a direkt geprintet.

The screenshot shows a Java code editor with the following code:

```
String test = "a";
for (; test.compareTo("aaa") == 0; test = test + "a") {
    System.out.print(test.length() + " ");
}
System.out.print(test);
```

Below the code, a tooltip asks "What is the output?". To the right, JavaDoc for `compareTo` is displayed:

`String compareTo(String anotherString)`
Compares this string lexicographically to the specified string. The result is a positive integer if this `String` object lexicographically follows the argument string. The result is zero if the strings are equal; `compareTo` returns 0 exactly when the `equals` (`Object`) method would return true.

This is the definition of lexicographic ordering. If two strings are different, then either they have different characters at some index that is a valid index for both strings, or their lengths are different, or both. If they have different characters at one or more index positions, let k be the smallest such index; then the string whose character at position k has the smaller value, as determined by using the `<` operator, lexicographically precedes the other string. In this case, `compareTo` returns the difference of the two character values at position k in the two strings -- that is, the value:

```
this.charAt(k)-anotherString.charAt(k)
```

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string. In this case, `compareTo` returns the difference of the lengths of the strings -- that is, the value:

```
this.length()-anotherString.length()
```

On the left, a sidebar contains the following text:

Nicht nur alphabetische Verschiebung, sondern auch Längenunterschiede werden verglichen (zumindest, wenn lexikografisch gleich), also "a".compareTo("aaa") = -2

Frage 87

Dienstag, 3. Dezember 2024 19:30

Given:

```
int iterations = 100;  
  
while (count < iterations) {  
    System.out.println("Iteration " + count);  
    count++;  
}
```

What is the result?

- A. Iteration plus an increasing number is printed 100 times.
- B. An error occurs during compilation.**
- C. Iteration plus an increasing number is printed 99 times.
- D. The program compiles and nothing is printed.

Vor der Ausführung der Schleife wird count nicht deklariert (und initialisiert).

Frage 88

Dienstag, 3. Dezember 2024 19:30

Given:

```
class MyClass {  
    private int var1 = 100;  
    public int var2 = 200;  
  
    public void doCalc() {  
        var1 = 100 * 2; // line n1  
        var2 = 200 * 2;  
    }  
}  
  
public class Test {  
    public static void main(String[] args) {  
        MyClass t = new MyClass();  
        t.doCalc();  
        System.out.println(t.var1 + " : " + t.var2); // line  
n2  
    }  
}
```

What is the result?

- A. A compilation error occurs at line n1.
- B. A compilation error occurs at line n2.
- C. 100 : 400
- D. 200 : 400

Mit t.var1 wird versucht von außen auf eine private-Variable zuzugreifen. Ein Getter wäre eine Alternative.

Frage 89

Dienstag, 3. Dezember 2024 19:30

Given:

```
int array[] = new int[3];
array[2] = 75;
array[0] = 25;
array[1] = 50;
```

```
List list = new ArrayList<>();
list.add(2, 75);
list.add(0, 25);
list.add(1, 50);
```

```
for (int aElement : array)
{ System.out.print(aElement + "% "); }
    System.out.println("");
    for (int lElement : list)
{ System.out.print(lElement + "% "); }
```

What is the result?

A.

25% 50% 75%
25% 50% 75%

B.

75% 25% 50%
75% 25% 50%

C.

A compilation error occurs

D.

A runtime exception is thrown.

Im Gegensatz zu normalen Arrays kann eine Arrayliste nicht in beliebiger Reihenfolge gefüllt werden.

Während Arrays feste Längen und Plätze haben (wie Kinositze), die beliebig besetzt werden können, sind Arraylisten flexibel in ihrer Länge, aber das heißt auch sie müssen "von unten nach oben stapelweise" gefüllt werden. Es dürfen beim Hinzufügen keine Lücken sein.

Frage 90

Mittwoch, 4. Dezember 2024 16:51

Given:

```
class Cartoon {  
    String name;  
  
    Cartoon(String name) {  
        this.name = name;  
    }  
  
    // line n1  
}  
  
public static void main(String[] args) {  
    Cartoon c1 = new Cartoon("tom");  
    Cartoon c2 = new Cartoon("tom");  
    // line n2  
}
```

Which modification enables the code to print true?

- A. At line n2, insert: System.out.println(c1.compareTo(c2));
- B. At line n1, override the hashCode() method from the Object class and, at line n2, insert : System.out.println(c1.compareTo(c2));
- C. At line n1, override the equals o method from the object class and, at line n2, insert : System.out.println(c1.equals(c2));
- D. At line n2, insert: System.out.println(c1.equals (c2));

Die Klasse Cartoon hat noch keine eigene compareTo-Methode. Teile der Objektklasse werden nur explizit vererbt, wenn überschrieben wird. A. geht nicht, weil der Override fehlt, B nennt eine andere Methode (hashCode()), D hat ein ähnliches Problem. Auch equals() braucht den Override. B ist die einzige Antwort bei der Override und Methodenaufruf zusammenpassen.