**Frage 1**

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

Automatisch generierte Beschreibung

Eine Variable darf $ heißen und der String (als lokale variable) wurde nicht initialisiert, ist also auch nicht implizit null, was hätte gedruckt werden können.

A: Ein Bezeichner (auch einer Klasse) darf mit \_ beginnen, nur nicht alleinstehen

B: Die Variable darf $ heißen und muss nicht initialisiert werden. (Gehört der Klasse)

C: String zwar nur deklariert, aber noch kein Fehler, solange kein Lesezugriff erfolgt.

D: $ ist implizit 0, weil es als statisch der Klasse gehört (aber auch Instanz wäre ok).

E: weil hier gelesen wird, wo noch nicht initialisiert wurde, auch nicht implizit.

F und G: Kommen nicht mehr in Frage, aufgrund des Compilerfehlers.

**Frage 3**

Ein Bild, das Text, Screenshot, Reihe, Schrift enthält.

Automatisch generierte Beschreibung

Es kommt nur C in Frage, weil alle anderen sich folgendermaßen ausschließen, bzw. C begründet wird durch nachfolgende Ausführung:

Erstes System.out.print

s1 und s1 beinhalten dasselbe Literal, dieses befindet sich auch im String Pool, da es als Literal hardgecodet wurde. Deshalb ergibt die erste Ausgabe (s1 == s2) **true.** (1x)

Zweites System.out.print

Strings werden per equals auf Wertegleichheit überprüft, nicht auf Selbigkeit des Objekts. Da s1 und s2 aufgrund derselben Referenz zum Stringpool aber bereits gleich sind, ist somit auch ihr Inhalt gleich, also **true** (2x)

Drittes System.out.print

Auch das Zusammensetzten aus (hardgecodeten) Einzelteilen, welches dann wieder einem anderen Literal im String-Pool entspricht, stellt eine Referenz und somit Gleichheit her. (betrifft hier die appends(„“) des String Builders. Leider stimmt der Abgleich aber nicht bzw. gibt die toString-Methode des StringBuilders ein anderes String-Objekt zurück als das aus dem Stringpool. Somit sind sie gleich aber nicht selbst. (**false)**

Viertes System.out.print

Die vierte Ausgabe arbeitet wie die zweite. Hier ist es egal, ob die Strings gleichen Ursprungs sind (Referenz), es zählt nur der Inhalt, also true. (3x)

Gründe für einen Kompilationsfehler lassen sich hier nicht finden.

**Frage 3**

Ein Bild, das Text, Elektronik, Screenshot, Display enthält.

Automatisch generierte Beschreibung

C, weil die Implementierung der public Interfacemethode als protected nicht passt.

D, weil von einer fehlerhaften Klasse geerbt wird (Puma), folglich ist auch Cougar falsch, um das zu beheben, muss wie oben genannt die getTailLength() in Puma public sein.

E, weil die Klasse Puma abstract ist und nicht initialisiert werden kann. Möglich hingegen wäre aber eine upcasting-Deklaration in Form von: Puma puma = new Cougar();  
Dies ist vergleichbar mit List<> list = new Arraylist()<>; (statt List<> list = new List()<>)

F trifft nicht zu, weil Zeile 11 ein legaler Overload wäre, wenn die Implementierung nicht bereits fehlerhaft wäre.

A, B und G kommen aufgrund der Fehler nicht in Frage.

**Frage XX**

Bild hier einfügen

Texte zu den Antworten.

Seitenumbruch