**Modultest (Simulation) Klasse JET AP11 Modul 2**

**Name**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pro jede richtig ausgefüllte Lücke bekommen Sie einen Punkt. Umseitig finden Sie eine Programmieraufgabe. Punkte für die praktische Aufgabe: siehe Aufstellung. Maximale Punkteanzahl bei diesem Test: **35**.

Benotung: Sehr gut (1): 100 % – 87 %, Gut (2): 86 % – 75 %,  
Befriedigend (3): 74 % – 63 %, Genügend (4): 62 % – 50 %,  
Nicht genügend (5): 49 % – 0 %

|  |  |
| --- | --- |
|  | Erreichte Punkte |
| 1. Die Methode \_\_\_\_\_\_\_\_\_\_\_\_\_\_ kann verwendet werden, um einen String in eine Zahl zu **konvertieren**. |  |
| 1. Die Bedingung in einer If-Verzweigung muss immer einen \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Wert liefern. |  |
| 1. In einer **If-Else-Kette** wird der \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Zweig ausgeführt, wenn keine vorherige Bedingung erfüllt ist. |  |
| 1. Um sicherzustellen, dass nach einem Case-Block nicht in den nächsten gesprungen wird, wird das Schlüsselwort \_\_\_\_\_\_\_\_\_\_\_\_\_\_ verwendet. |  |
| 1. Falls kein Case zutrifft, wird der \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Block ausgeführt. |  |
| 1. Eine \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Schleife wird mindestens einmal ausgeführt, da die **Bedingung am Ende** geprüft wird. |  |
| 1. Eine \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Schleife wird oft für eine bekannte Anzahl an Durchläufen verwendet |  |
| 1. Auf ein Element eines Arrays greift man mit dem \_\_\_\_\_\_\_\_\_\_\_\_\_\_ zu. |  |
| 1. Der **Index** eines Arrays beginnt standardmäßig bei \_\_\_\_\_\_\_\_\_\_\_\_\_\_. |  |
| 1. **Verweistypen** werden im \_\_\_\_\_\_\_\_\_\_\_\_\_\_ gespeichert. |  |
| 1. Beispiele für **Werttypen** sind \_\_\_\_\_\_\_\_\_\_\_\_\_\_ und \_\_\_\_\_\_\_\_\_\_\_\_\_\_. |  |
| 1. Beispiele für **Verweistypen** sind \_\_\_\_\_\_\_\_\_\_\_\_\_\_ und \_\_\_\_\_\_\_\_\_\_\_\_\_\_. |  |
| 1. Speicher im **Heap** wird durch den \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ automatisch verwaltet. |  |
| 1. Optionale Parameter müssen in der Methodensignatur einen \_\_\_\_\_\_\_\_\_\_\_\_\_\_ besitzen. |  |
| 1. Eine abgeleitete Klasse **erbt** von einer \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Klasse. |  |
| 1. Eine Klasse kann man mit dem Schlüsselwort \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **versiegeln**. |  |
| 1. Der Konstruktor einer Basisklasse kann in einer abgeleiteten Klasse mit \_\_\_\_\_\_\_\_\_\_\_\_\_\_ aufgerufen werden. |  |

**Bitte wenden!**

**Praktische Aufgabe**

Erstellen Sie eine kleine Anwendung zur Verwaltung von Tieren. Nutzen Sie dabei die Prinzipien der Vererbung in C#.

Die Ausgabe könnte, wie folgt aussehen:  
Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Anforderungen:**

1. (4 Punkte)  
   Erstellen Sie eine Basisklasse **Tier** mit folgenden Eigenschaften:
   * ArtType\* **Art**
   * string **Name**
   * int **Alter**
   * double **Gewicht**
   * Methode **void ZeigeInfo()**, die alle Eigenschaften auf der Konsole ausgibt.

\* Artpype ist ein enum (Hund, Katze)

Erstellen sie auch einen Konstruktor, der alle die Eigenschaften initialisiert.

1. (6 Punkte)  
   Erstellen Sie zwei abgeleitete Klassen:
   * **Hund**, das zusätzlich die Eigenschaften bool **KannBellen** und string **Rasse** besitzt.
   * **Katze,** das zusätzlich die Eigenschaft bool **HatKrallen** besitzt.

Auch hier brauchen Sie (jeweils für Klasse Hund und Klasse Katze) je einen Konstruktor, der auch den Konstruktor, der der Basis-Klasse aufruft.

1. (8 Punkte)  
   In der **Main()-**Methode sollen:
   * Ein Array Hund[] **hunde** mit mindestens zwei Hunden erstellt werden.
   * Ein Array Hund[] **katzen** mit mindestens zwei Katzen erstellt werden.
   * Eine Benutzereingabe soll es ermöglichen, Katzen oder Hunde auszugeben. Nutzen Sie dazu ein **switch**-Statement.
   * Mithilfe einer **for**-Schleife soll unter dem entsprechenden **Case** die **ZeigeInfo()-**Methode für jedes Objekt im Array aufgerufen werden.

Der **Quellcode** ist als .zip-Archiv **an Ihren Trainer** zu schicken.

Viel Erfolg bei der Umsetzung!