**CS 1302 – Sample Test 1a Handout**

*Tear this page off to use for reference* ● *Turn this page in with test.*

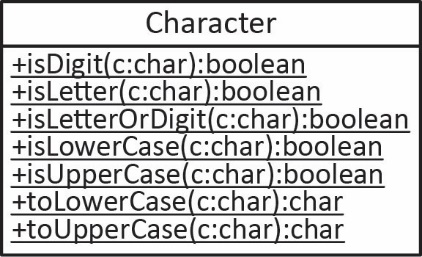
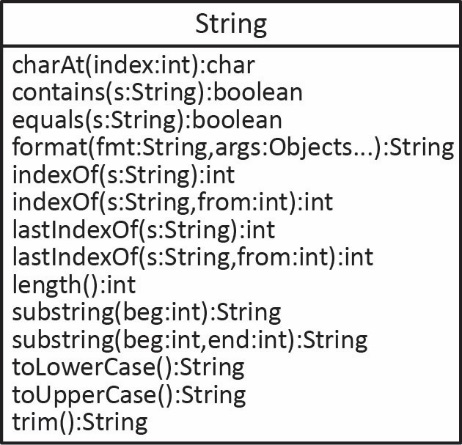
**All problems on the test use this information:**

You will write a *Dpg* (can abbreviate: *Emp*)class with the following characteristics:

* has *name* and *age* (int) properties.
* A constructor that accepts both the *name* and *age.*
* A constructor that accepts a *name* only, and sets the *age* to 1using best practices.
* A getter and setter for the *age* property only. (you do not need to write a getter and setter for *name*; however, you can assume they exist).

It should also have these methods:

* *bark* – accepts an integer and return a string “bark” concatenated the number of times specified by the integer. For example: *bark(3)* would return: “bark bark bark”.
* *increateAge* – accepts a percent (in decimal form) and increases the age of this dog by that percent. (**This solution does not provide a solution for this method)**
* *dogNameAbbreviation* (can abbreviate: *gNA*)– Returns the dog name’s abbreviation which is the first three characters of the dog’s name. You can assume a dog’s name is 3 or more characters long. (**This solution does not provide a solution for this method)**
* *barkAt* – accepts another *Dog* and returns a string like this: “thisDogName barks at anotherDogName”. For example, if this dog is named *fido* and the input dog is named *leo,* then the method will return: “fido barks at leo”.
* *countOlder* – accepts an array of dogs and returns the number of dogs in the array that this dog is older than.

 **

|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 1a** | **Name:** |  |
|  |  | Print: FirstName LastName |

***Closed Book Test. No Notes Allowed. Write answers on the test. This is longer than an actual test would be.***

1. (~50 points) Write the *Dog* class described on the handout.
2. (~20 points) Write a single line of code that:

|  |  |
| --- | --- |
| 1. Creates a *Dog* using the first constructor. Call it *d1*. |  |

|  |  |
| --- | --- |
| 1. Creates a *Dog* using the second constructor. Call it *d2*. |  |

|  |  |
| --- | --- |
| 1. Changes the *age* of *d1* to 5 |  |

|  |  |
| --- | --- |
| 1. Has *d1* bark 4 times storing the result in a variable. |  |

|  |  |
| --- | --- |
| 1. Has *d1* bark at *d2* storing the result in a variable. |  |

|  |  |
| --- | --- |
| 1. Suppose you have dogs: *d5* and *d6.* Define an array to hold these dogs and put the dogs in the array. You can do this in one line or several. |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| 1. Calls the *countOlder* methodon *d1*, passing the array of dogs from the previous step. The result should be stored in a variable. |  |

1. Suppose you have a class, *DogUtilities.* This class has no instance variables, it just has methods that accept arrays of *Dog*s.

**Note: on an actual test, you will write 2 or 3 methods like the ones below, additional ones are provided for practice.**

1. (~10 points) Write a method for this class, *getAverageAge,* that accepts an array of *Dog* objectsand returns the average age of the dogs in the array. (**The solution does not provide a solution for this method)**
2. (~10 points) Write a method for this class, *getOldestAge,* that accepts an array of *Dog* objectsand returns the age of the oldest *Dog*. (**The solution does not provide a solution for this method)**
3. (~10 points) Write a method for this class, *getOldestDog,* that accepts an array of *Dog* objectsand returns the *Dog* that is the oldest. age of the dogs in the array. (**The solution does not provide a solution for this method)**
4. (~10 points) Write a method for this class, *getAges,* that accepts an array of *Dog* objectsand returns a new array of the age of each dog in the input array. (**The solution does not provide a solution for this method)**
5. (~10 points) Write a method for this class, *getEveryOther,* that accepts an array of *Dog* and returns a new array of *Dog* that contains every other *Dog* from the array that was passed to the method. An example is below; however, note that it shows only the dog name’s, when it really represents actual *Dog* objects.

|  |  |
| --- | --- |
| Example 1 | Example 2 |
| Input: [fido, snoopy, leo, chaps] | Input: [fido, snoopy, leo, chaps, zorro] |
| Output: [fido, leo] | Output: [fido, leo, zorro] |

1. (~10 points) Write a method for this class, *createMutants,* that accepts an array of *Dog* and returns a new array of *Dog*. This array has a new dog for each input dog where the name of each dog is the same as the input except that the first and last characters are removed. For example:

|  |  |
| --- | --- |
| Input Array | Output Array |
| [name:”Fido”, age=3 | [name:”id”, age=3 |
| name:”Snoopy”, age=4 | name:”noop”, age=4 |
| name:”Leo”, age=2] | name:”e”, age=2] |

1. (~6 points) Consider this sample piece of code:

double len = 14.823423489;

String product = “crackers”;

Int quantity = 4

String msg = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

System.out.println(msg)

Complete the line of code to create a formatted string that when printed, displays:

“Quantity of 4 crackers, where each has length of 14.823”

Note: this is just an example. Your code should work no matter what the values of the variables are. You must use the variables themselves, not their values in the answer.