|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 3** | **Name:** |  |
| No electronics-phone, smart watch, notes, *etc.* allowed. |  | Print: FirstName LastName |

***Write all code using best practices. Reference the handout on last page***

1. Write the abstract *Animal* class including the 2 highlighted methods. You can assume the other methods are implemented.

* The *feed* method is abstract.
* Override the *equals* method so that it will return *true* if two *Animals* have the same name, and *false* otherwise.

1. Write the entire *Bear* class.

* The *feed* method should reduce *hunger* by 1, and return the (new) *hunger.* However, *hunger c*annot fall below zero. Thus, if *hunger* is zero, a call to *feed* will not change *hunger* and simply return zero.

1. Write the entire *MommaBear* class.

* If number of cubs is not specified in the constructer, it should be set to the default of 2.
* The *feed* method should reduce *hunger* by 1 for itself and each of its cubs, and return the (new) *hunger,* provide the hunger is not reduced below zero. If it would reduce *hunger* below zero, then *hunger* is reduced to zero. For example: (a) if a bear has a hunger of 5 and 3 cubs, then when its feed method is called, *hunger* is reduced to 1 which is returned. (b) if a bear has a hunger of 2 and 3 cubs, then when its feed method is called, *hunger* would potentially be reduced to -2, but instead is reduced to 0 which is returned.

1. Write the entire *Zoo* class.
   1. You can assume the class has a no-arg constructor which you do not need to write.
   2. The class has an *ArrayList* of *Animals*.
   3. *addAnimal* accepts an *Animal* and adds it to the list of *Animals* provide there doesn’t already exist an *Animal* in the *Zoo* with the same *name*. In this case, it also returns *true*; otherwise, *false*.
   4. *addAnimal2* simply adds the argument to the list. Hint: this is very simple. Just add the animal!
   5. *getNumAnimals()* returns the number of *Animals* in the *Zoo*.
   6. *getAnimal(index:int)* returns the *Animal* at *index,* if the *index* is valid. Otherwise, it returns *null.*
   7. *getAnimal(name:String)* returns the *Animal* whose *name* exactly matches the input *name*. Otherwise, it returns *null.*
   8. *removeAnimal(index:int)* removes the *Animal* at *index* from the list of *Animals* and returns the removed *Animal.* You can assume the index is valid.
   9. *removeAnimal(name:String)* removes the *Animal* whose *name* exactly matches the input *name*. Otherwise, it returns *null.*
   10. *getMommaBears(numCubs:int)* returns an *ArrayList* of all the *MommaBears* with at least as many cubs as specified in the parameter. For example, *getMommaBears(2)* would return Ursa with 2 cubs and Barda with 3 cubs, but not Sammy with 1 cub.
   11. The feedAll method calls the feed method for all animals at the zoo and returns the sum total of the returned values.
2. Write the following snippets of code:
3. Make a *Bear* object and store it in a variable.
4. Make two *MommaBear* objects using different constructors, storing both in different variables.
5. Make a *Zoo* object and store it in a variable.
6. Add all three *Animals* created above to the *Zoo*.
7. Assume lots of animals have been added to the zoo. Get the 8th animal from the *Zoo* and store it in a variable.
8. Get the *Animal* with name *Urg* from the *Zoo* and store it in a variable.
9. Remove the 5th animal in the zoon and store the result in a variable
10. Remove the animal with name, “Quola” from the *Zoo* and store the result in a variable.
11. Call a method that returns a list of the *MammaBears* from the *Zoo* with at least 3 cubs and be sure to define a variable to hold the return from this method.
12. Call the *feedAll* method and catch the result in a variable.
13. Write several lines of code to build a string with each *Animals* *toString* (assume there is one) appended together with line breaks. Hint: loop over all the *Animals* in the *Zoo*.
14. Write the following snippets of code:
15. Modify the *Person* class below such that you could sort an *ArrayList<Person>* on *name*.

* You are not to write code to sort an ArrayList, you are modifying the *Player* class so that a list could be sorted.
* You do not need to rewrite the class, just make the modifications out to the side and draw an arrow where the go in the class.
* Hint: there are 3 separate things you need to do.

**public** **class** Person {

**private** String name;

**private** **int** age;

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **int** getAge() {

**return** age;

}

}

1. Suppose you have an *ArrayList<Person>* named *persons* for which a number of *Person* objects have been added. Write a line of code to sort this list on *name*.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**CS 1302 – Handout, Sample Test 3**

Use the information below for problems 1-5. As prompted on the test, **you will write the code for all the items that are highlighted, using best practices.** Assume any non-highlighted members are present (*i.e.* you do not need to write them).

Descriptions of the methods are provided with the problems in the test.

