**CS 1302 – Sample Test 2 Handout**

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

****The information below will be used for problems 1-2.

***Dog* Class**

**You can assume the *Dog* class is already written, except for this method which you will write:**

* *bulkUp(amount:double)* – Increases the dog’s *weight* by *amount*.

***WolfDog* Class**

**You will write most of the *WolfDog* (can abbreviation: WD) class**, which is a subclass of *Dog.* It requires following members:

1. *toughness* instance variable.
2. *WolfDog(name:String, weight:double, toughness:int)* – Constructor.
3. *WolfDog(name:String, weight:double)* – Constructor, in this case, the toughness is set to 1. Use best practices.
4. *getPower():double* – Returns the *weight* of the dog multiplied by the *toughness*.
5. **You do not need to write the *getToughness* method; however, you can assume it exists.**

***Person* Class**

**You will write most of the *Person* class.** This class requires the following members:

1. An array of size=20of *Dog* objects (you may NOT use an *ArrayList*)
2. *addDog(d:Dog)* – Adds *d* to *dogs* if there is available space in the array.
3. *addDogs(newDogs:Dog[])* – add all the *newDogs* to the *dogs* array. You can assume *newDogs* is full.
4. *getDog(i:int):Dog* – Returns the dog at *i* if *i* is valid and *null* otherwise.
5. *getDogWithName(name:String)* – Returns the dog whose name is *name,* if found, and *null* otherwise.
6. *remDog(i:int):Dog* – Returns and removes the dog at *i* (you can assume *i* is valid).
7. *totalDogPower():double* – Returns the total power of all the dogs (and subclasses) added together.
8. *averageWolfDogPower():double* – Returns the average power of all the wolf dogs.
9. *totalWolfDogToughness():int* – Returns the total toughness of all the wolf dogs added together.
10. *leastWeightDog():Dog* – Returns the dog with least weight
11. *maxWeight():double –* Returns the weight of the dog with largest weight.

You can assume the *Person* class has:

* a no-arg constructor that does nothing.
* a *getNumDogs* method that returns the number of dogs in the *dogs* array.

|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 2** | **Name:** |  |
|  |  | Print: FirstName LastName |

***Closed Book Test. No Notes Allowed. Write answers on the test.***

1. (60-70 points) Write the code described on the handout.
	1. Write the method only for the *Dog* class.
	2. Write the *WolfDog* class as described.
	3. Write the *Person* class as described.
2. (15-20 points) Write a line of code that:
3. Creates a *Dog* (any name and weight)

|  |
| --- |
|  |

1. Creates a *WolfDog* with toughness equal to 4 (any name and weight)

|  |
| --- |
|  |

1. Creates a *Person*.

|  |
| --- |
|  |

1. Adds one of the dogs from above to the person.

|  |
| --- |
|  |

*For the rest of the questions, assume many dogs have been added to the person.*

1. Use the appropriate *Person* method to get the 5th dog and store the result in a variable.

|  |
| --- |
|  |

1. Use the appropriate *Person* method to get the total power of all the dogs added together and store the result in a variable.

|  |
| --- |
|  |

1. Use the appropriate *Person* method to get dog with the least weight and store the result in a variable.

|  |
| --- |
|  |

1. Suppose you have references to 2 dogs: *d1* & *wd2.* Define an array, *newDogs*, to hold these two dogs and put them in the array. Write 1-3 lines to accomplish this.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

1. Use the appropriate *Person* method to add the dogs in *newDogs* (part *h* above) to the person.

|  |
| --- |
|  |

1. Suppose you have a reference to a *Person, p* that has some number of dogs. Write a snippet of code to loop through the person and print the weight of each dog with one decimal.

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

1. Consider classes *A* and *B* shown in the table below. What output do each of these object creations produce?

|  |  |  |
| --- | --- | --- |
|  | Code | Output |
| a. | A a1 = new B(); |  |
| b. | A a2 = new B(22); |  |
| c. | A a3 = new B("Q"); |  |

|  |  |
| --- | --- |
| class A { public A() { this("M"); }  public A(String s) { System.out.print("L" + s); }} | class B extends A { public B() { this("N"); }  public B(int num) { System.out.print("P"); } public B(String s) { super(s); System.out.print("O"); }} |

1. Consider the class diagram and references shown below. For each of the *Assignments*, determine whether the assignment compiles (valid) or does not compile (invalid)

|  |  |  |
| --- | --- | --- |
| **Class Diagram** | **References** | **Assignments** |
|  | A a = new A();B b = new B();C c = new C();D d = new D(); | 1. B b1 = a; Compiles or Doesn’t Compile
2. D d1 = b; Compiles or Doesn’t Compile
3. A a1 = b; Compiles or Doesn’t Compile
4. C c1 = d; Compiles or Doesn’t Compile
5. B b2 = d; Compiles or Doesn’t Compile
6. A a2 = d; Compiles or Doesn’t Compile
7. B b3 = c; Compiles or Doesn’t Compile
8. A a3 = c; Compiles or Doesn’t Compile
 |

* 1. Consider the partial *StackOfPersons* class below. In the box below, write the *push* & *pop* methods for this stack.

**public** **class** StackOfPersons {

 **private** Person[] persons;

 **private** **int** size = 0;

 **public** StackOfPersons(**int** capacity) {

 persons = **new** Person[capacity];

 }

 **public** **boolean** isEmpty() {

 **return** size == 0;

 }

|  |
| --- |
|  |

}

* 1. Suppose you have two *Person* objects, *p1* and *p2.* Write a snippet of code to
		1. create a *StackOfPersons* with capacity 50(abbreviate *SOP*),

|  |
| --- |
|  |

* + 1. push the two persons onto the stack,

|  |
| --- |
|  |

|  |
| --- |
|  |

* + 1. pop the stack, storing the value that is returned in a variable.

|  |
| --- |
|  |