|  |  |  |
| --- | --- | --- |
| **CS 1302 – Sample Test 2** | **Name:** | **KEY** |
|  |  | 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.

**public** **void** bulkUp(**double** amount) {

 weight += amount;

}

* 1. Write the *WolfDog* class as described.

**public** **class** WolfDog **extends** Dog {

 **private** **int** toughness;

 **public** WolfDog(String name, **double** weight, **int** toughness) {

 **super**(name,weight);

 **this**.toughness = toughness;

 }

 **public** WolfDog(String name, **double** weight) {

 **this**(name, weight, 1.0);

 }

 @Override

 **public** **double** getPower() {

 **return** toughness \* weight;

 }

 **public** **int** getToughness() {

 **return** toughness;

 }

}

* 1. Write the *Person* class as described.

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

 **private** **int** numDogs;

 **private** Dog[] dogs = **new** Dog[10];

 **public** Person() {

 }

 **public** **int** getNumDog() {

 **return** numDogs;

 }

 **public** **void** addDog(Dog dog) {

 **if**(numDogs < dogs.length) {

 dogs[numDogs] = dog;

 numDogs++;

 }

 }

**public** **void** addDogs(Dog[] newDogs) {

 **for**(Dog d: newDogs) {

 addDog(d);

 }

}

**public** **void** addDogs2(Dog[] newDogs) {

 **for**(**int** i = 0; i < newDogs.length; i++) {

 addDog(newDogs[i]);

 }

}

 **public** Dog getDog(**int** i) {

 **if**(i>=0 && i< numDogs) {

 **return** dogs[i];

 }

 **return** **null**;

 }

 **public** Dog dogWithName(String name) {

 **for** (**int** i = 0; i < numDogs; i++) {

 **if** (dogs[i].getName().equals(name)) {

 **return** dogs[i];

 }

 }

 **return** **null**;

 }

 **public** Dog removeDog(**int** i) {

 **if**(i>= 0 && i<numDogs) {

 Dog returnDog = dogs[i];

 **for** (**int** j=i+1; j<numDogs; j++) {

 dogs[j-1] = dogs[j];

 }

 numDogs--;

 **return** returnDog;

 }

 **return** **null**;

 }

 **public** **double** totalDogPower() {

 **double** tp = 0.0;

 **for** (**int** i =0; i<numDogs; i++) {

 tp += dogs[i].getPower();

 }

 **return** tp;

 }

 **public** **double** totalWolfDogToughness() {

 **double** sum = 0.0;

 **for**(**int** i = 0; i<numDogs; i++) {

 Dog d = dogs[i];

 **if** (d **instanceof** WolfDog) {

 WolfDog wd = (WolfDog)d;

 sum += wd.getToughness();

 }

 }

 **return** sum;

 }

**public** **double** averageWolfDogPower() {

 **double** sum = 0.0;

 **int** count = 0;

 **for**(**int** i = 0; i<numDogs; i++) {

 Dog d = dogs[i];

 **if** (d **instanceof** WolfDog) {

 sum += d.getPower();

 count++;

 }

 }

 **if**(count > 1) {

 **return** sum/count;

 }

 **return** sum;

 }

 **public** Dog leastWeightDog() {

 Dog leastWeightDog = dogs[0];

 **for**(**int** i = 0; i<numDogs; i++) {

 Dog d = dogs[i];

 **if** (d.getWeight() < leastWeightDog.getWeight()) {

 leastWeightDog = d;

 }

 }

 **return** leastWeightDog;

 }

 **public** **double** leastWeight() {

 **double** leastWeight = dogs[0].getWeight();

 **for**(**int** i = 0; i<numDogs; i++) {

 **double** currentWeight = dogs[i].getWeight();

 **if** (currentWeight < leastWeight) {

 leastWeight = currentWeight;

 }

 }

 **return** leastWeight;

 }

}

1. (15-20 points) Write a line of code that:

Dog d = **new** Dog("Fido", 20.5);

WolfDog wd2 = **new** WolfDog("Wolfie", 30.5, 10);

Person p = **new** Person();

p.addDog(d);

p.addDog(wd2);

Dog d2 = p.getDog(4);

**double** tp = p.totalDogPower();

Dog d3 = p.leastWeightDog();

Dog[] newDogs = {d, wd2};

p.addDogs(newDogs);

**for** (**int** i = 0; i < p.getNumDogs(); i++) {

 String wt = String.*format*("%.1f", p.getDog(i).getWeight());

 System.***out***.println(wt);

}

1. Trace the following constructor chains based on the classes below and provide the output:

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

1. Consider the following class hierarchy diagram. Determine which lines of code are valid and which are invalid.
2. B b1 = a; Compiles or Doesn’t Compile
3. D d1 = b; Compiles or Doesn’t Compile
4. A a1 = b; Compiles or Doesn’t Compile
5. C c1 = d; Compiles or Doesn’t Compile
6. B b2 = d; Compiles or Doesn’t Compile
7. A a2 = d; Compiles or Doesn’t Compile
8. B b3 = c; Compiles or Doesn’t Compile
9. 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;

 }

|  |
| --- |
| **public** **void** push(Person d) { **if**(size<persons.length) { persons[size++] = d; }}**public** Person pop() { **if**(!isEmpty()) { **return** persons[--size]; } **return** **null**;} |

}

* 1. Suppose you have two *Person* objects, *p1* and *p2.* Write a snippet of code to

StackOfPersons sop = **new** StackOfPersons(50);

sop.push(p1);

sop.push(p2);

Person p = sop.pop();