**44-542 Object Oriented Programming Exam 02 Part 02 (40 Points) KEY**

**Do NOT use lambda expressions on this exam.**

All of the questions on this part of the exam refer to the classes on the Dog handout. If you do not have this handout, ask your instructor for a copy.

1. (10 pts) Note that **IllegalDogAgeException** extends **RuntimeException** and therefore is an unchecked exception and does not have to be advertised.

In the **DogList** class, rewrite method **addDog** so that an **IllegalDogAgeException** is thrown if the age of the dog passed as an argument is less than zero. Otherwise, the dog is added to the array list **dogs**. The message **Dog age is invalid** should be included as an argument to the exception.

Write your answer in the space below. Write the complete method, including the method header, but write ***only*** method **addDog**.

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

**if(dog.getAge() < 0) {**

**throw new IllegalDogAgeException("Dog age is invalid");**

**} else {**

**dogs.add(dog);**

**}**

**}**

1. (10 pts) In the **DogDriver** class, add code to the **while** loop to catch the **IllegalDogAgeException** that you threw in the previous question on this exam.

Example: Suppose **dogs.txt** has this data

**Leonard 6**

**Lacie -3**

**Eve 8**

**Bas -1**

The output produced by the driver would be:

**Skipping Lacie -- age is less than 0**

**dogs.IllegalDogAgeException: Dog age is invalid**

**Skipping Bas -- age is less than 0**

**dogs.IllegalDogAgeException: Dog age is invalid**

Your code must work for any valid data set. The output produced must match exactly the output shown above.

Write your answer in the space below. Write ***only*** the **while** loop. ***Do not change the three lines currently in the* while** ***loop***.

**while (in.hasNext()) {**

**try {**

**dogName = in.next();**

**dogAge = in.nextInt();**

**myDogs.addDog(new Dog(dogName, dogAge));**

**} catch(IllegalDogAgeException ex) {**

**System.out.println("Skipping " + dogName +**

**" -- age is less than 0\n" + ex);**

**}**

**}**

1. (5 pts) Note that the **Dog** class implements the **Comparable<Dog> i**nterface. However, the code for the **compareTo** method is missing. In the space below, write the missing code. Dogs should be compared on the basis of their name. This means that if a list of dogs is sorted using the natural order as supplied by this **compareTo** method, the resulting list will be in alphabetical order by name

**@Override**

**public int compareTo(Dog otherDog) {**

**return this.name.compareTo(otherDog.name);**

**}**

1. (5 pts) In the **DogList** class, provide the code for method **naturalSort**. This method sorts the dogs in the array list named dogs according to their natural order.

**public void naturalSort() {**

**Collections.sort(dogs);**

**}**

1. (10 pts) In the **DogList** class, provide the code for method **sortByAge**. This method sorts the dogs in the array list in ascending order by age. You must use a **Comparator** for this method, and you must create an anonymous class. Do not create an external class.

**public void sortByAge() {**

**Collections.sort(dogs, new Comparator<Dog>() {**

**@Override**

**public int compare(Dog dog01, Dog dog02) {**

**return dog01.getAge() - dog02.getAge();**

**}**

**});**

**}**