Practice Exercise #23: Manage Animals

http://www.comp.nus.edu.sg/~cs1020/4_misc/practice.html

Objective:

- Inheritance with Generics
- Substitutability

Task Statement

A business has two establishments: Monyet Zoo and Burung Bird Park. A program has already been written to list the names of animals in the Zoo, but you are to extend this to the Bird Park, with some additional requirements.

The existing programs are found in the skeleton file. **Zoo.java** describes a list of animals, along with some types of animals:

```
class Animal { ... }
class Monkey extends Animal { ... }
class Chimp extends Animal { ... }

class Zoo <E> {

   private List<E> animals;

   public Zoo() {
      // Substitutability in action (1) -
      //   Supertype reference to subtype object
      animals = new ArrayList<E>();
   }
   public void addAnimal(E newAnimal) {
      animals.add(newAnimal);
   }
   @Override
   public String toString() {
      // ArrayList.toString() is invoked at runtime
      return animals.toString();
   }
}
```

Notice how a reference-typed variable may have a different datatype as the object it refers to. A `List<E>` reference can point to an `ArrayList<E>` object. If both types implement a method with the same signature, can you recall which method is executed when the program is run?

ManageAnimals.java contains the driver class, which reads in (specie, name) pairs:

```
public class ManageAnimals {

   private Zoo<Animal> zoo; // zoo only contains Monkeys or Chimps
   /* private ____ birdPark; */

   public ManageAnimals() {
      zoo = new Zoo<Animal>();
      /* birdPark = new ____(); */
   }

   public void listAnimals () {
      System.out.println("Zoo: " + zoo);
      /* System.out.println("Bird Park: " + birdPark); */
   }

   public void addAnimal(String specie, String name) {// INCOMPLETE
      switch (specie) {
         // Substitutability in action (2) -
         //    Animal expected, Monkey (or Chimp) provided
         case "Monkey":
            zoo.addAnimal(new Monkey(name));
            return;
         case "Chimp":
            zoo.addAnimal(new Chimp(name));
            return;
      }
   }

   public static void main(String[] args) {
      ManageAnimals manager = new ManageAnimals();
      Scanner sc = new Scanner(System.in);
      while (sc.hasNext())
         manager.addAnimal(sc.next(), sc.next());
      sc.close();
      manager.listAnimals();
   }
}
```

There are 99 species of Bird, so it is not feasible to create 100 additional classes to describe them. Therefore:

- Create a **Bird** class in **Zoo.java** with attributes `specie` and `name`
- Fill in the missing data types in the driver class in **ManageAnimals.java**
- Complete the addAnimal() method to add all other animals to the Bird Park

Remember to code incrementally. Compile and test your program at each step.

Submit BOTH **Zoo.java** and **ManageAnimals.java**.

Sample Input #1

```
Monkey monyet
Kingfisher k
Chick tweety
Chimp panz
Monkey irbaboon
```

Sample Output #1

```
Zoo: [Monkey:monyet, Chimp:panz, Monkey:irbaboon]
Bird Park: [Kingfisher:k, Chick:tweety]
```

Sample Input #2

```
Duck daisy
Roadrunner ppp
Duck daffy
Bird burung
```

Sample Output #2

```
Zoo: []
Bird Park: [Duck:daisy, Roadrunner:ppp, Duck:daffy, Bird:burung]
```

Think…

Do you see how generics are *useful* here?
Do you see how generics are *limited* here?
Can the Zoo class handle anything that applies to all animals?