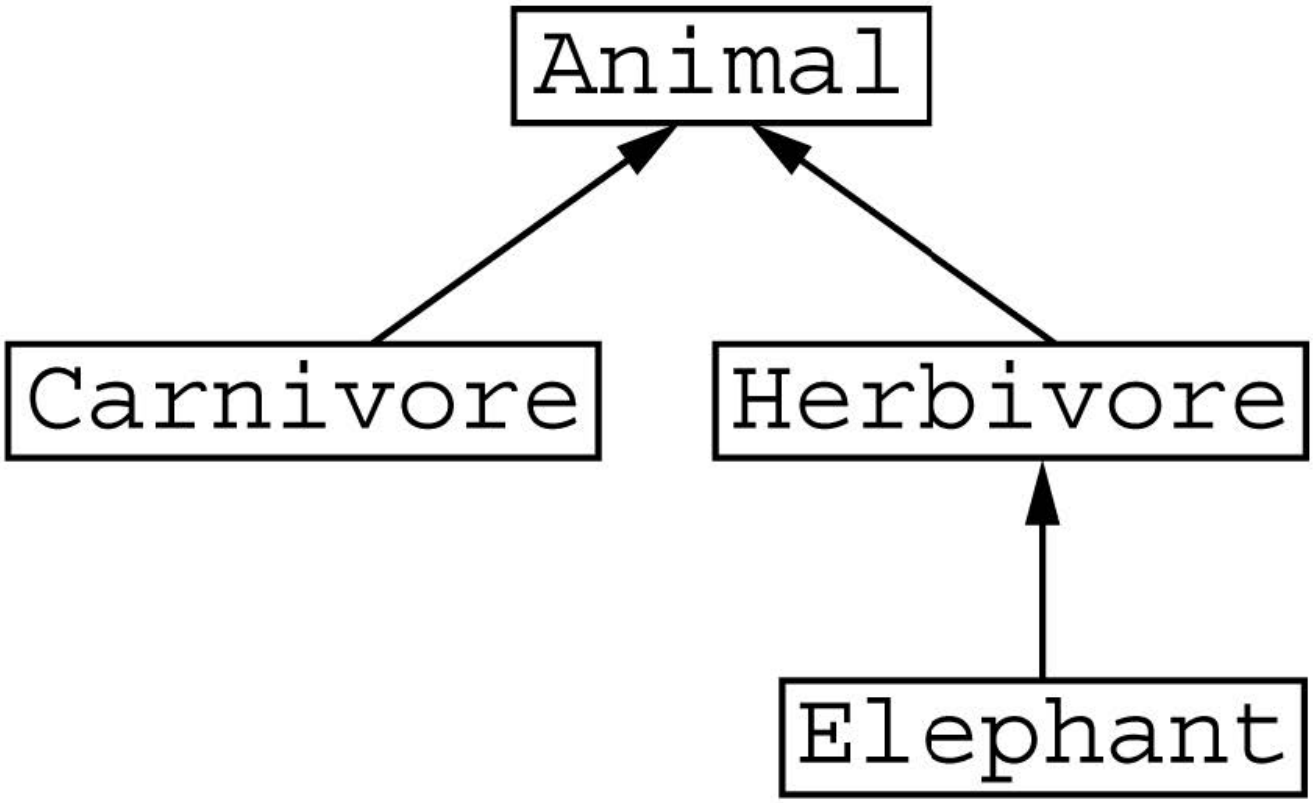
Animal FRQ

SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Assume that the classes listed in the Java Quick Reference have been imported where appropriate. Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods will not receive full credit.

A set of classes using inheritance is used to represent animals observed at a wildlife sanctuary. A portion of the class hierarchy is shown in the following diagram.



All `Animal` objects have the following attributes.

A `String` variable indicating whether the animal is a carnivore or a herbivore
A `String` variable representing the animal species (e.g., lion, giraffe, zebra)
A `String` variable representing the name of the individual animal (e.g., Lisa, Gary, Percy)
The `Animal` class also contains a `toString` method that indicates the state of an animal.

The following table shows the intended behavior of the `Animal` class.

Statement	Result
`Animal lisa = new Animal("carnivore", "lion", "Lisa");`	A new `Animal` object is created.
`lisa.toString();`	The string "`Lisa the lion is a carnivore`" is returned.

(a) Write the complete `Animal` class. Your implementation must meet all specifications and conform to the behavior shown in the table.

The `Herbivore` class is a subclass of `Animal`. The `Herbivore` class does not contain any attributes or methods other than those found in the `Animal` class.

The constructor to the `Herbivore` class accepts two parameters for the species and name of the herbivore. The constructor passes those parameters along with the string literal `"herbivore"` to the `Animal` class to construct the object.

The following table shows the intended behavior of the `Herbivore` class.

Statement	Result
`Herbivore gary = new Herbivore("giraffe", "Gary");`	A new `Herbivore` object is created.
`gary.toString();`	The string "`Gary the giraffe is a herbivore`" is returned.

(b) Write the complete `Herbivore` class. Your implementation must meet all specifications and conform to the behavior shown in the table.

The `Elephant` class is a subclass of `Herbivore`. The `Elephant` class contains one additional attribute not found in `Herbivore`: a `double` variable representing the length of the elephant's tusks, in meters.

The constructor to the `Elephant` class accepts two parameters for the name and tusk length of the elephant. The constructor passes those parameters along with the string literal `"elephant"` to the `Herbivore` class to construct the object.

The following table shows the intended behavior of the `Elephant` class.

Statement	Result
`Elephant percy = new Elephant("Percy", 2.0);`	A new `Elephant` object is created.
`percy.toString();`	The string "`Percy the elephant is a herbivore with tusks 2.0 meters long`" is returned.

(c) Write the complete `Elephant` class. Your implementation must meet all specifications and conform to the behavior shown in the table.