Unit 9 Progress Check: FRQ

1. 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.

The following `Book` class is used to represent books and print information about each book. Each `Book` object has attributes for the book title and for the name of the book's author.

```
public class Book
{
     private String title;
     private String author;

     public Book(String t, String a)
     {
         title = t;
         author = a;
     }

     public void printBookInfo()
     {
         System.out.print(title + ", written by " + author);
     }
}
```

(a) The `PictureBook` class is a subclass of the `Book` class that has one additional attribute: a `String` variable named `illustrator` that is used to represent the name of the illustrator of a picture book. The `PictureBook` class also contains a `printBookInfo` method to print the title, writer, and illustrator of a picture book.

Consider the following code segment.

```
PictureBook myBook = new PictureBook("Peter and Wendy", "J.M. Barrie",
"F.D. Bedford");
myBook.printBookInfo();
```

The code segment is intended to print the following output.

```
Peter and Wendy, written by J.M. Barrie and illustrated by F.D. Bedford
```

Complete the `PictureBook` class below. Your implementation should conform to the example above.

```
public class PictureBook extends Book
```

Unit 9 Progress Check: FRQ

Consider the following books.

A book titled *Frankenstein*, written by Mary Shelley
A picture book titled *The Wonderful Wizard of Oz*, written by L. Frank Baum and illustrated by W.W. Denslow

The following code segment is intended to represent the two books described above as objects referenced by variables `book1` and `book2`, respectively, and add them to the `ArrayList myLibrary`.

```
ArrayList<Book> myLibrary = new ArrayList<Book>();
/* missing code */
myLibrary.add(book1);
myLibrary.add(book2);
```

(b) Write a code segment that can be used to replace `/* missing code */` so that `book1` and `book2` will be correctly declared and initialized. Assume that class `PictureBook` works as intended, regardless of what you wrote in part (a).

The `BookListing` class is used to generate a descriptive listing for a book. The `BookListing` constructor takes a `Book` object and a `double` value as parameters and uses them to print information about the book, along with its price.

Assume that the objects referenced by `book1` and `book2` were created as specified in part (b). The following table demonstrates the intended behavior of the `BookListing` class using objects referenced by `book1` and `book2`.

Code Segment	Result Printed
`BookListing listing1 = new BookListing(book1, 10.99); listing1.printDescription();`	`Frankenstein, written by Mary Shelley, $10.99`
`BookListing listing2 = new BookListing(book2, 12.99); listing2.printDescription();`	`The Wonderful Wizard of Oz, written by L. Frank Baum and illustrated by W.W. Denslow, $12.99`

(c) Complete the `BookListing` class below. Your implementation should conform to the examples. Assume that class `PictureBook` works as intended, regardless of what you wrote in part (a).

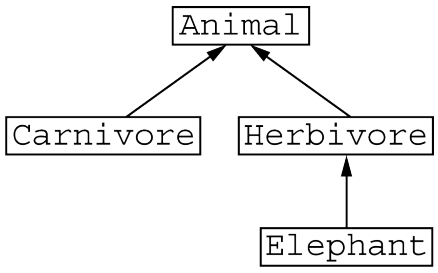
```
public class BookListing
```

Unit 9 Progress Check: FRQ

2. 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.