```java
/*
 * Programmer:  Chad Philip Johnson
 * Date Created:  Thursday, November 15th, 2012
 * Date of Last Modification:  Thursday, November 15th, 2012
 *
 * Description:
 * Barn.class contains subroutines for adding and removing instances of Animals.class to and from its someAnimals array.
 */

import java.util.*;
import java.io.Serializable;

 /**
 * Barn.class contains subroutines for adding and removing instances of Animals.class to and from its someAnimals array.
 *
 * @author Chad Philip Johnson
 * @version 1.0
 */

public class Barn implements Serializable   {

    Animal[] someAnimals;
    int intNumberOfAnimals;

    /**
      * Default constructor:
      * Initialize variables for a new instance of Barn.class.
      */

    public Barn()   {

        someAnimals         = new Animal[3];
        intNumberOfAnimals  = 0;

    }

    /**
     * Overloaded constructor:
     * Initialize variables for a new instance of Barn.class and establish the number of instances of type Animal can be stored
     within the local
     * array someAnimals.
     * @param intArrayLength The size of the array someAnimals.
     */

    public Barn( int intArrayLength )   {

        someAnimals         = new Animal[intArrayLength];
        intNumberOfAnimals  = 0;

    }
```

```
50
51      /**
52       * Add an animal to the partially filled array someAnimals and increment the value of the helper variable intNumberOfAnimals
53       *
54       * @param animalToAdd The animal to store in the barn.
55       */
56
57      public void addAnimal( Animal animalToAdd ) {
58
59          // NOTE:  Array should be packed before calling this method
60
61          // Add the animal the currently available element in the array
62          // which is represented by the previous value for
63          // intNumberOfAnimals, then increment intNumberOfAnimals by one
64          someAnimals[intNumberOfAnimals++]   = animalToAdd;
65
66      }
67
68      /**
69       * Return the instance of Animal.class stored in the someAnimals array at the supplied index.
70       *
71       * @param intIndex The index of someAnimals to return.
72       * @return The instance of Animal.class stored at the supplied index.
73       */
74
75      public Animal getAnimal( int intIndex ) {
76
77          return someAnimals[intIndex];
78
79      }
80
81      /**
82       * Put an instance of Animal.class into the partially filled array someAnimals and increment the helper variable
         intNumberOfAnimals.
83       *
84       * @param animalToPut The instance of Animal.class to put in the array someAnimals.
85       * @param intIndex The index of someAnimals in which to store the supplied instance of Animals.class.
86       */
87
88      public void putAnimal( Animal animalToPut, int intIndex )   {
89
90          // NOTE:  Verify "null" at the desired index location before
91          // calling this function
92
93          someAnimals[intIndex]   = animalToPut;
94          intNumberOfAnimals++;
95
96      }
97
98      /**
```

```
99          * Remove the instance of Animal.class stored in the someAnimals array at the supplied index.
100         *
101         * @param intIndex The index of someAnimals to set to null.
102         */
103
104        public void removeAnimal( int intIndex )    {
105
106            someAnimals[intIndex] = null;
107            packBarn();
108
109        }
110
111        /**
112         * Remove the instance of Animal.class stored in the someAnimals array at the supplied index; includes the option to pack
            the array.
113         *
114         * @param intIndex The index of someAnimals to return.
115         * @param blnPackAnimals Boolean statement to instruct whether the partially filled array someAnimals should be packed after
            the removal.
116         * @return The instance of Animal.class stored at the supplied index.
117         */
118
119        public Animal removeAnimal( int intIndex, boolean blnPackAnimals )  {
120
121            Animal removedAnimal;
122
123            removedAnimal   = someAnimals[intIndex];
124            someAnimals[intIndex] = null;
125
126            intNumberOfAnimals--;
127
128            if( blnPackAnimals )    {   packBarn(); }
129
130            return removedAnimal;
131
132        }
133
134        /**
135         * Return the maximum number of animals that can be stored in the partially filled array someAnimals.
136         * @return The number of animals in the partially filled array someAnimals.
137         */
138
139        public int getMaxNumberOfAnimals()  {
140
141            return someAnimals.length;
142
143        }
144
145        /**
146         * Return the total number of animals stored in the partially filled array someAnimals.
```

```
147         *
148         * @return The number of animals in the partially filled array someAnimals.
149         */
150
151        public int getCurrentNumberOfAnimals()  {
152
153            return intNumberOfAnimals;
154
155        }
156
157        /**
158         * Pack the partially filled array someAnimals.
159         */
160
161        public void packBarn()  {
162
163            int intAnimalIndex      = 0;
164            int intPackIndex        = -1;
165            boolean blnContinuePacking  = true;
166
167            while( blnContinuePacking ) {
168
169                blnContinuePacking  = false;
170
171                while( intAnimalIndex < (someAnimals.length - 1) )  {
172
173                    // Always assign the last encountered null space to intPackIndex to increase efficiency of packing algorithm
174                    if( ( someAnimals[intAnimalIndex] == null ) && ( someAnimals[(intAnimalIndex + 1)] != null ) )  {
175
176                        if( intPackIndex != -1 )    {
177
178                            someAnimals[intPackIndex] = someAnimals[(intAnimalIndex + 1)];
179                            someAnimals[(intAnimalIndex + 1)]   = null;
180
181                            intPackIndex = (intAnimalIndex + 1);
182
183                        } else {
184
185                            someAnimals[intAnimalIndex] = someAnimals[(intAnimalIndex + 1)];
186                            someAnimals[(intAnimalIndex + 1)]   = null;
187
188                            intPackIndex    = (intAnimalIndex + 1);
189
190                        }
191
192                        blnContinuePacking = true;
193
194                    } else if( ( intPackIndex == -1 ) && ( someAnimals[intAnimalIndex] == null ) )  {
195
196                        intPackIndex    = intAnimalIndex;
```

```
                }

                intAnimalIndex++;

            }

            intAnimalIndex  = 0;
            intPackIndex    = -1;

        }

    }

    /**
     * Invoke the toString() methods of all instances of Animal.class stored in the partially filled array someAnimals.
     * @return Details of each instance of Animal.class stored in the array someAnimals.
     */

    public String toString()    {

        String strBarnAnimals   = "";

        int intAnimalCount  = 0;
        for( Animal animal : someAnimals )  {

            intAnimalCount++;

            if( animal != null )    {

                strBarnAnimals  += "\t[" + Integer.toString( intAnimalCount ) + "] " + animal;

            } else {

                strBarnAnimals  += "\t[" + Integer.toString( intAnimalCount ) + "] Vacant\n";

            }

        }

        return strBarnAnimals;

    }

}
```