

OBrienLukeAssignment3.java

```

1 //----- Imports
2 import java.util.ArrayList;
3 import java.util.Scanner;
4 import java.io.FileReader;
5 import java.io.FileNotFoundException;
6
7
8 /*
9  * Name:           Luke O'Brien
10 * Class Name:      Data Structure and Algorithms
11 * Class Section:   002
12 * Assignment Number: 3
13 * Due Date:        2/12/2020 (Wednesday)
14 *
15 * General Description:
16 * This program imports a text file with insect data. It then goes on
17 * to parse the data and store it into a special data structure
18 * made from a class hierarchy that uses implementations of interfaces
19 * and abstract classes to properly store and organize all the data.
20 * The program then goes and prints some highlighted data to the user.
21 */
22
23
24 //----- Main Class
25 public abstract class OBrienLukeAssignment3
26 {
27
28     public static void main(String[] args) throws FileNotFoundException //-----
29     {
30         Main Method
31         {
32             FileReader importedFile = new FileReader("insects.txt");
33             Scanner parser = new Scanner(importedFile);
34
35             int arraySize = 0;
36             if(parser.hasNextInt())
37                 arraySize = parser.nextInt();
38             parser.nextLine();
39
40             Insect[] insects = new Insect[arraySize];
41
42             //----- Start For Loop --- Runs through array assigning values and types to
43             each spot
44             for(int x=0; x<arraySize; x++)
45             {
46                 //----- Parses all the information from textFile
47                 String type = parser.next();
48                 String name = parser.next();
49                 int pollinatorRank = parser.nextInt();
50                 int builderRank = parser.nextInt();
51                 int predatorRank = parser.nextInt();
52                 int decomposeRank = parser.nextInt();
53                 parser.hasNextLine();
54
55                 //----- Takes information and puts it into proper data structure
56                 if(type.toLowerCase().equals("h"))
57                     insects[x] = new HoneyBee("Honey Bee", name, pollinatorRank, builderRank);
58                 else if(type.toLowerCase().equals("a"))

```

O'BrienLukeAssignment3.java

```

56         insects[x] = new Ant("Ant", name, builderRank, predatorRank, decomposeRank);
57     else if(type.toLowerCase().equals("p"))
58         insects[x] = new PrayingMantis("Praying Mantis", name, predatorRank);
59     else if(type.toLowerCase().equals("l"))
60         insects[x] = new LadyBug("Ladybug", name, pollinatorRank, predatorRank);
61 }
62 //----- END For Loop
63
64 parser.close();
65
66 //----- Printing the Output
67
68 //----- Prints out all the insects that don't decompose
69 System.out.println("The Insects that DON'T help with Decomposition
are:\n-----");
70 for(Insect x : findDoNotDecompose(insects))
71 {
72     System.out.println(x.getName()+" is A "+x.getType()+" and does not help with
Decomposition.");
73     System.out.println("\n"+x.purpose()+"\n");
74     displayAbilities(findDoNotDecompose(insects).get(findDoNotDecompose(insects).index
Of(x)));
75     System.out.println("\n-----");
76 }
77 //-----Prints out the most Able insect
78 System.out.println("\n\nThe Insect With the most
Abilities:\n-----");
79 System.out.println("The Winner is "+insects[findMostAble(insects)].getName()+" the
"+insects[findMostAble(insects)].getType());
80 System.out.println("\n" + insects[findMostAble(insects)].purpose() + "\n");
81 displayAbilities(insects[findMostAble(insects)]);
82 }
83
84 //----- Finds insects that do not decompose
85 public static ArrayList<Insect> findDoNotDecompose(Insect[] insects)
86 {
87     /*
88      * Creates an array list then goes through the given array checking for
89      * anything that is NOT an instance of 'Decomposer.' If it isn't, then
90      * it will add it to the array list
91      */
92
93     ArrayList<Insect> returnArray = new ArrayList<>();
94
95     for(int x=0; x<insects.length; x++)
96     {
97         if(!(insects[x] instanceof Decomposer))
98             returnArray.add(insects[x]);
99     }
100     return returnArray;
101 }
102
103 //----- Finds the most 'able' insect out of the array
104 public static int findMostAble(Insect[] insects)
105 {
106     /*
107      * takes all the values of each object in the array, then adds them

```

OBrienLukeAssignment3.java

```

108     * to determine which of the objects has the highest 'score'
109     * It then returns the index of that highest scoring object
110     */
111
112     int indexNum = 0;
113     int refScore = 0;
114
115     for(int x=0; x<insects.length; x++)
116     {
117         if(insects[x] instanceof HoneyBee)
118         {
119             if(refScore < ( ((HoneyBee)insects[x]).getBuilderRank() +
120 ((HoneyBee)insects[x]).getPollinateRank() ) )
121             {
122                 indexNum = x;
123                 refScore = ((HoneyBee)insects[x]).getBuilderRank() +
124 ((HoneyBee)insects[x]).getPollinateRank();
125             }
126         }
127         else if(insects[x] instanceof PrayingMantis)
128         {
129             if(refScore < ( ((PrayingMantis)insects[x]).getPredatorRank() ) )
130             {
131                 indexNum = x;
132                 refScore = ((PrayingMantis)insects[x]).getPredatorRank();
133             }
134         }
135         else if(insects[x] instanceof Ant)
136         {
137             if(refScore < ( ((Ant)insects[x]).getBuilderRank() +
138 ((Ant)insects[x]).getPredatorRank() + ((Ant)insects[x]).getDecomposeRank() ) )
139             {
140                 indexNum = x;
141                 refScore = ((Ant)insects[x]).getBuilderRank() +
142 ((Ant)insects[x]).getPredatorRank() + ((Ant)insects[x]).getDecomposeRank();
143             }
144         }
145         else if(insects[x] instanceof LadyBug)
146         {
147             if(refScore < ( ((LadyBug)insects[x]).getPollinateRank() +
148 ((LadyBug)insects[x]).getPredatorRank() ) )
149             {
150                 indexNum = x;
151                 refScore = ((LadyBug)insects[x]).getPollinateRank() +
152 ((LadyBug)insects[x]).getPredatorRank();
153             }
154         }
155     }
156
157     return indexNum;
158 }
159
160 //----- Prints out all the abilities of a given insect
161 public static void displayAbilities(Insect insects)
162 {
163     /*
164     * Checks to see what type of bug is being passed through, then

```

O'BrienLukeAssignment3.java

```

159     * prints out its respected abilities through object casting
160     */
161
162     System.out.println("-----");
163     if(insects instanceof LadyBug)
164     {
165         System.out.println("Pollination Level: " + ((LadyBug)insects).getPollinateRank());
166         System.out.println("Predatory Level: " + ((LadyBug)insects).getPredatorRank());
167     }
168     else if(insects instanceof HoneyBee)
169     {
170         System.out.println("Pollination Level: " +
171 ((HoneyBee)insects).getPollinateRank());
172         System.out.println("Building Level: " + ((HoneyBee)insects).getBuilderRank());
173     }
174     else if(insects instanceof Ant)
175     {
176         System.out.println("Building Level: " + ((Ant)insects).getBuilderRank());
177         System.out.println("Predatory Level: " + ((Ant)insects).getPredatorRank());
178         System.out.println("Decomposition Level: " + ((Ant)insects).getDecomposeRank());
179     }
180     else if(insects instanceof PrayingMantis)
181     {
182         System.out.println("Predatory Level: " +
183 ((PrayingMantis)insects).getPredatorRank());
184     }
185     System.out.println("-----");
186 }
187 //----- Interfaces
188 interface Pollinator
189 {
190     int getPollinateRank();
191 }
192 interface Builder
193 {
194     int getBuilderRank();
195 }
196 interface Predator
197 {
198     int getPredatorRank();
199 }
200 interface Decomposer
201 {
202     int getDecomposeRank();
203 }
204
205 //----- *** Class Hierarchy ***
206
207 //----- Insect Super class
208 abstract class Insect
209 {
210     private String type;
211     private String name;
212
213     public void setType(String type)

```

```

214     {
215         this.type = type;
216     }
217     public void setName(String name)
218     {
219         this.name = name;
220     }
221     public String getType()
222     {
223         return type;
224     }
225     public String getName()
226     {
227         return name;
228     }
229     public abstract String purpose(); //returns the set purpose of selected insect
230 }
231
232 //----- HoneyBee SubClass
233 class HoneyBee extends Insect implements Pollinator, Builder
234 {
235     private int pollinateRank;
236     private int builderRank;
237
238     HoneyBee(){
239         //Default Constructor
240     }
241
242     HoneyBee(String type, String name, int pollinateRank, int builderRank)
243     {
244         setType(type);
245         setName(name);
246         this.pollinateRank = pollinateRank;
247         this.builderRank = builderRank;
248     }
249
250     @Override
251     public String purpose() //from abstract class 'Insect'
252     {
253         return "I'm popular for producing honey but I also pollinate 35% of the
crops!\nWithout me, 1/3 of the food you eat would not be available!";
254     }
255
256     @Override
257     public int getPollinateRank() //from interface "Pollinator"
258     {
259         return pollinateRank;
260     }
261
262     @Override
263     public int getBuilderRank() //From interface "Builder"
264     {
265         return builderRank;
266     }
267 }
268
269 //----- Praying Mantis SubClass

```

```

270 class PrayingMantis extends Insect implements Predator
271 {
272     private int predatorRank;
273
274     PrayingMantis(){
275         //Default Constructor
276     }
277
278     PrayingMantis(String type, String name, int predatorRank)
279     {
280         setType(type);
281         setName(name);
282         this.predatorRank = predatorRank;
283     }
284
285     @Override
286     public String purpose() //from abstract class "Insect"
287     {
288         return "I'm an extreme predator quick enough to catch a fly.\nRelease me in a garden
and I'll eat beetles, grasshoppers, crickets and even pesky moths.";
289     }
290
291     @Override
292     public int getPredatorRank() //from interface "Predator"
293     {
294         return predatorRank;
295     }
296 }
297
298 //----- Ant SubClass
299 class Ant extends Insect implements Builder, Decomposer, Predator
300 {
301     private int builderRank;
302     private int decomposeRank;
303     private int predatorRank;
304
305     Ant(){
306         //Default Constructor
307     }
308
309     Ant(String type, String name, int builderRank, int predatorRank, int decomposeRank)
310     {
311         setType(type);
312         setName(name);
313         this.builderRank = builderRank;
314         this.predatorRank = predatorRank;
315         this.decomposeRank = decomposeRank;
316     }
317
318     @Override
319     public String purpose() //from abstract class "Insect"
320     {
321         return "Don't squash me, I'm an ecosystem engineer!\nMe and my 20 million friends
accelerate decomposition of dead wood,\naerate soil, improve drainage, and eat insects like
ticks and termites!";
322     }
323

```

O'BrienLukeAssignment3.java

```

324     @Override
325     public int getBuilderRank() //from interface "Builder"
326     {
327         return builderRank;
328     }
329
330     @Override
331     public int getDecomposeRank() //from interface "Decomposer"
332     {
333         return decomposeRank;
334     }
335
336     @Override
337     public int getPredatorRank() //from interface "Predator"
338     {
339         return predatorRank;
340     }
341 }
342
343 //-----LadyBug SubClass
344 class LadyBug extends Insect implements Pollinator, Predator
345 {
346     private int pollinateRank;
347     private int predatorRank;
348
349     LadyBug(){
350         //Default Constructor
351     }
352
353     LadyBug(String type, String name, int pollinateRank, int predatorRank)
354     {
355         setType(type);
356         setName(name);
357         this.pollinateRank = pollinateRank;
358         this.predatorRank = predatorRank;
359     }
360
361     @Override
362     public String purpose() //from abstract class "Insect"
363     {
364         return "Named after the Virgin Mary, I'm considered good luck if I land on you!\nI'm a
pest control expert eating up to 5,000 plant pests during my life span.";
365     }
366
367     @Override
368     public int getPollinateRank() //from interface "Pollinator"
369     {
370         return pollinateRank;
371     }
372
373     @Override
374     public int getPredatorRank() //from interface "Predator"
375     {
376         return predatorRank;
377     }
378 }

```