```java
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;

/**
 * Name: Zane Emerick
 * Class: CS 1450 Section 001
 * Assignment #3
 * Due: Feb 13, 2020
 *
 * Description: Create an insect abstract class and multiple subclasses that
 * represent various types of insects. These insects need to use interfaces
 * to represent their different abilities. Next, read text from a file and
 * categorize it into the various insect types and create methods that can
 * categorize them based on whether the insect helps with decomposition or
 * not. After that, create a method that can determine the insect with the
 * highest overall stats. Finally, display all of this data to the user.
 */

public class EmerickZaneAssignment3 {
    public static void main(String[] args) throws IOException {
        File file = new File("insects.txt");
        Scanner reader = new Scanner(file);

        int arrayLength = reader.nextInt();
        Insect[] insects = new Insect[arrayLength];

        //read info and create new insect
        for(int i = 0; i < insects.length; i++) {
            String type = reader.next().substring(0, 1);
            String name = reader.next();
            int pollinateAbility = reader.nextInt();
            int buildAbility = reader.nextInt();
            int predatorAbility = reader.nextInt();
            int decomposeAbility = reader.nextInt();

            switch(type) {
                case "h":
                    insects[i] = new HoneyBee(pollinateAbility, buildAbility,
name);
                    insects[i].setType("honey bee");
                    break;
                case "l":
                    insects[i] = new LadyBug(predatorAbility,
pollinateAbility, name);
                    insects[i].setType("lady Bug");
                    break;
                case "a":
                    insects[i] = new Ant(predatorAbility, buildAbility,
decomposeAbility, name);
                    insects[i].setType("ant");
                    break;
                case "p":
                    insects[i] = new PrayingMantis(predatorAbility, name);
                    insects[i].setType("praying mantis");
                    break;
            }
        }
        //writing portion of assignment
        ArrayList<Insect> nonDecomposers = findDoNotDecompose(insects);
```

```
58          int mostAbilities = findMostAble(insects);
59
60          System.out.println("Insects that do not help with decomposition:");
61          System.out.println("-------------------------------------------");
62          for(int i = 0; i < nonDecomposers.size(); i++) {
63              System.out.printf("%n%s is a %s and does not help with
    decomposition%n", nonDecomposers.get(i).getName(),
    nonDecomposers.get(i).getType());
64              System.out.println(nonDecomposers.get(i).purpose());
65              displayAbilities(nonDecomposers.get(i));
66          }
67
68          System.out.println("\nInsect with the most abilities:");
69          System.out.println("------------------------------\n");
70          System.out.printf("The winner is %s the %s%n",
    insects[mostAbilities].getName(), insects[mostAbilities].getType());
71          System.out.println(insects[mostAbilities].purpose());
72          displayAbilities(insects[mostAbilities]);
73
74          reader.close();
75      }
76
77      /**
78       * parse the list for insects who do not help decompose matter.
79       * @param insects an array containing all of the insects
80       * @return an ArrayList containing every insect who is not a decomposer
81       */
82      public static ArrayList<Insect> findDoNotDecompose(Insect[] insects) {
83          ArrayList<Insect> nonDecomposers = new ArrayList<Insect>();
84
85          for(int i = 0; i < insects.length; i++) {
86              if(!(insects[i] instanceof Decomposer)) {
87                  nonDecomposers.add(insects[i]);
88              }
89          }
90          return nonDecomposers;
91      }
92
93      /**
94       * find the insect with the highest overall stats
95       * @param insects  an array containing all of the insects
96       * @return the index of the insect with the highest total stats
97       */
98      public static int findMostAble(Insect[] insects) {
99          int topAbilityNumber = 0;
100         int mostAbleIndex = 0;
101         for(int i = 0; i < insects.length; i++) {
102             int currentAbilityNumber = 0;
103             if(insects[i] instanceof HoneyBee) {
104                 currentAbilityNumber = ((HoneyBee)insects[i]).pollinate();
105                 currentAbilityNumber += ((HoneyBee)insects[i]).build();
106
107             } else if(insects[i] instanceof LadyBug) {
108                 currentAbilityNumber = ((LadyBug)insects[i]).pollinate();
109                 currentAbilityNumber += ((LadyBug)insects[i]).predator();
110
111             } else if(insects[i] instanceof Ant) {
112                 currentAbilityNumber = ((Ant)insects[i]).build();
113                 currentAbilityNumber += ((Ant)insects[i]).decompose();
114                 currentAbilityNumber += ((Ant)insects[i]).predator();
```

```
115
116             } else if(insects[i] instanceof PrayingMantis) {
117                 currentAbilityNumber +=
((PrayingMantis)insects[i]).predator();
118             }
119
120             if(currentAbilityNumber > topAbilityNumber) {
121                 topAbilityNumber = currentAbilityNumber;
122                 mostAbleIndex = i;
123             }
124         }
125
126         return mostAbleIndex;
127     }
128     /**
129      * Display the various abilities of a specified insect
130      * @param insect the insect to be displayed
131      */
132     public static void displayAbilities(Insect insect) {
133         if(insect instanceof HoneyBee) {
134             System.out.println("Pollinating ability: " +
((HoneyBee)insect).pollinate());
135             System.out.println("Building ability: " +
((HoneyBee)insect).build());
136         } else if (insect instanceof LadyBug) {
137             System.out.println("Pollinating ability: " +
((LadyBug)insect).pollinate());
138             System.out.println("Predator ability: " +
((LadyBug)insect).predator());
139         } else if (insect instanceof Ant) {
140             System.out.println("Building ability: " + ((Ant)insect).build());
141             System.out.println("Predator ability: " +
((Ant)insect).predator());
142             System.out.println("Decomposing ability: " +
((Ant)insect).decompose());
143         } else if (insect instanceof PrayingMantis) {
144             System.out.println("Predator ability: " +
((PrayingMantis)insect).predator());
145         }
146     }
147 }
148
149 interface Pollinator {
150     public abstract int pollinate();
151 }
152
153 interface Builder {
154     public abstract int build();
155 }
156
157 interface Predator {
158     public abstract int predator();
159 }
160
161 interface Decomposer {
162     public abstract int decompose();
163 }
164
165
166 abstract class Insect {
```

```
    private String name;
    private String type;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getType() {
        return type;
    }

    abstract String purpose();
}

class HoneyBee extends Insect implements Pollinator, Builder {
    private int pollinateAbility;
    private int buildAbility;

    public HoneyBee(int pollinateAbility, int buildAbility, String name) {
        this.pollinateAbility = pollinateAbility;
        this.buildAbility = buildAbility;
        setName(name);
        setType("HoneyBee");
    }

    @Override
    public String purpose() {
        return "I'm popular for producing honey but I also pollinate 35% of the crops!" +
        " Without me, 1/3 of the food you eat would not be available!";
    }

    @Override
    public int build() {
        return buildAbility;
    }

    @Override
    public int pollinate() {
        return pollinateAbility;
    }
}

class LadyBug extends Insect implements Pollinator, Predator {
    private int predatorAbility;
    private int pollinateAbility;

    public LadyBug(int predatorAbility, int pollinateAbility, String name) {
        this.predatorAbility = predatorAbility;
        this.pollinateAbility = pollinateAbility;
        setName(name);
        setType("LadyBug");
```

```
    }

    @Override
    public String purpose() {
        return "Named after the Virgin Mary, I'm considered good luck if I land on you!" +
        " I'm a pest control expert eating up to 5,000 plant pests during my life span.";
    }

    @Override
    public int predator() {
        return predatorAbility;
    }

    @Override
    public int pollinate() {
        return pollinateAbility;
    }
}

class Ant extends Insect implements Builder, Predator, Decomposer {
    private int buildAbility;
    private int predatorAbility;
    private int decomposeAbility;

    public Ant(int predatorAbility, int buildAbility, int decomposeAbility, String name) {
        this.predatorAbility = predatorAbility;
        this.buildAbility = buildAbility;
        this.decomposeAbility = decomposeAbility;
        setName(name);
        setType("Ant");
    }

    @Override
    public String purpose() {
        return "Don't squash me, I'm an ecosystem engineer! Me and my 20 million friends" +
        " accelerate decomposition of dead wood, aerate soil, improve drainage, and eat " +
        "insects like ticks and termites!";
    }

    @Override
    public int build() {
        return buildAbility;
    }

    @Override
    public int predator() {
        return predatorAbility;
    }

    @Override
    public int decompose() {
        return decomposeAbility;
    }
}

```

```
class PrayingMantis extends Insect implements Predator {
    private int predatorAbility;

    public PrayingMantis(int predatorAbility, String name) {
        this.predatorAbility = predatorAbility;
        setName(name);
        setType("PrayingMantis");
    }

    @Override
    public String purpose() {
        return "I'm an extreme predator quick enough to catch a fly. Release me in a garden" +
        "and I'll eat beetles, grasshoppers, crickets and even pesky moths.";
    }

    @Override
    public int predator() {
        return predatorAbility;
    }
}
```