```java
//--------------------------------------- Imports
import java.util.ArrayList;
import java.util.Scanner;
import java.io.FileReader;
import java.io.FileNotFoundException;


/*
 * Name:                   Luke O'Brien
 * Class Name:             Data Structure and Algorithms
 * Class Section:          002
 * Assignment Number:      3
 * Due Date:               2/12/2020 (Wednesday)
 *
 * General Description:
 * This program imports a text file with insect data. It then goes on
 * to parse the data and store it into a special data structure
 * made from a class hierarchy that uses implementations of interfaces
 * and abstract classes to properly store and organize all the data.
 * The program then goes and prints some highlighted data to the user.
 */


//--------------------------------------- Main Class
public abstract class OBrienLukeAssignment3
{

    public static void main(String[] args) throws FileNotFoundException //-------------------- Main Method
    {
        FileReader importedFile = new FileReader("insects.txt");
        Scanner parser = new Scanner(importedFile);

        int arraySize = 0;
        if(parser.hasNextInt())
            arraySize = parser.nextInt();
        parser.nextLine();

        Insect[] insects = new Insect[arraySize];

        //--------------- Start For Loop --- Runs through array assigning values and types to each spot
        for(int x=0; x<arraySize; x++)
        {
            //----- Parses all the information from textFile
            String type = parser.next();
            String name = parser.next();
            int pollinatorRank = parser.nextInt();
            int builderRank = parser.nextInt();
            int predatorRank = parser.nextInt();
            int decomposeRank = parser.nextInt();
            parser.hasNextLine();

            //----- Takes information and puts it into proper data structure
            if(type.toLowerCase().equals("h"))
                insects[x] = new HoneyBee("Honey Bee", name, pollinatorRank, builderRank);
            else if(type.toLowerCase().equals("a"))
```

```java
                insects[x] = new Ant("Ant", name, builderRank, predatorRank, decomposeRank);
            else if(type.toLowerCase().equals("p"))
                insects[x] = new PrayingMantis("Praying Mantis", name, predatorRank);
            else if(type.toLowerCase().equals("l"))
                insects[x] = new LadyBug("Ladybug", name, pollinatorRank, predatorRank);
        }
        //--------------- END For Loop

        parser.close();

        //-------------------- Printing the Output

        //-------------------- Prints out all the insects that don't decompose
        System.out.println("The Insects that DON'T help with Decomposition are:\n--------------------------------------------------");
        for(Insect x : findDoNotDecompose(insects))
        {
            System.out.println(x.getName()+" is A "+x.getType()+" and does not help with Decomposition.");
            System.out.println("\""+x.purpose()+"\"\n");
            displayAbilities(findDoNotDecompose(insects).get(findDoNotDecompose(insects).indexOf(x)));
            System.out.println("\n----------------------------------");
        }
        //----------Prints out the most Able insect
        System.out.println("\n\n\nThe Insect With the most Abilities:\n--------------------------------------------------");
        System.out.println("The Winner is "+insects[findMostAble(insects)].getName()+" the "+insects[findMostAble(insects)].getType());
        System.out.println("\"" + insects[findMostAble(insects)].purpose() + "\"");
        displayAbilities(insects[findMostAble(insects)]);
    }

    //-------------------- Finds insects that do not decompose
    public static ArrayList<Insect> findDoNotDecompose(Insect[] insects)
    {
        /*
         * Creates an array list then goes through the given array checking for
         * anything that is NOT and instance of 'Decomposer." If it isn't, then
         * it will add it to the array list
         */

        ArrayList<Insect> returnArray = new ArrayList<>();

        for(int x=0; x<insects.length; x++)
        {
            if(!(insects[x] instanceof Decomposer))
                returnArray.add(insects[x]);
        }
        return returnArray;
    }

    //-------------------- Finds the most 'able' insect out of the array
    public static int findMostAble(Insect[] insects)
    {
        /*
         * takes all the values of each object in the array, then adds them
         * to determine which of the objects has the highest 'score'
         * It then returns the index of that highest scoring object
         */

        int indexNum = 0;
        int refScore = 0;

        for(int x=0; x<insects.length; x++)
        {
            if(insects[x] instanceof HoneyBee)
            {
                if(refScore < ( ((HoneyBee)insects[x]).getBuilderRank() + ((HoneyBee)insects[x]).getPollinateRank() ) )
                {
                    indexNum = x;
                    refScore = ((HoneyBee)insects[x]).getBuilderRank() + ((HoneyBee)insects[x]).getPollinateRank();
                }
            }
            else if(insects[x] instanceof PrayingMantis)
            {
                if(refScore < ( ((PrayingMantis)insects[x]).getPredatorRank() ) )
                {
                    indexNum = x;
                    refScore = ((PrayingMantis)insects[x]).getPredatorRank();
                }
            }
            else if(insects[x] instanceof Ant)
            {
                if(refScore < ( ((Ant)insects[x]).getBuilderRank() + ((Ant)insects[x]).getPredatorRank() + ((Ant)insects[x]).getDecomposeRank() ) )
                {
                    indexNum = x;
                    refScore = ((Ant)insects[x]).getBuilderRank() + ((Ant)insects[x]).getPredatorRank() + ((Ant)insects[x]).getDecomposeRank();
                }
            }
            else if(insects[x] instanceof LadyBug)
            {
                if(refScore < ( ((LadyBug)insects[x]).getPollinateRank() + ((LadyBug)insects[x]).getPredatorRank() ) )
                {
                    indexNum = x;
                    refScore = ((LadyBug)insects[x]).getPollinateRank() + ((LadyBug)insects[x]).getPredatorRank();
                }
            }
        }

        return indexNum;
    }

    //-------------------- Prints out all the abilities of a given insect
    public static void displayAbilities(Insect insects)
    {
        /*
         * Checks to see what type of bug is being passed through, then
         * prints out its respected abilities through object casting
         */

        System.out.println("-----");
        if(insects instanceof LadyBug)
        {
            System.out.println("Pollination Level: " + ((LadyBug)insects).getPollinateRank());
            System.out.println("Predatory Level: " + ((LadyBug)insects).getPredatorRank());
        }
        else if(insects instanceof HoneyBee)
        {
            System.out.println("Pollination Level: " +
((HoneyBee)insects).getPollinateRank());
            System.out.println("Building Level: " + ((HoneyBee)insects).getBuilderRank());
        }
        else if(insects instanceof Ant)
        {
            System.out.println("Building Level: " + ((Ant)insects).getBuilderRank());
            System.out.println("Predatory Level: " + ((Ant)insects).getPredatorRank());
            System.out.println("Decomposition Level: " + ((Ant)insects).getDecomposeRank());
        }
        else if(insects instanceof PrayingMantis)
        {
            System.out.println("Predatory Level: " +
((PrayingMantis)insects).getPredatorRank());
        }
        System.out.println("-----");
    }
}

//---------------------------------------- Interfaces
interface Pollinator
{
    int getPollinateRank();
}
interface Builder
{
    int getBuilderRank();
}
interface Predator
{
    int getPredatorRank();
}
interface Decomposer
{
    int getDecomposeRank();
}

//-------------------------------------------------- *** Class Hierarchy ***

//-------------------- Insect Super class
abstract class Insect
{
    private String type;
    private String name;

    public void setType(String type)
```

```java
    {
        this.type = type;
    }
    public void setName(String name)
    {
        this.name = name;
    }
    public String getType()
    {
        return type;
    }
    public String getName()
    {
        return name;
    }
    public abstract String purpose(); //returns the set purpose of selected insect
}

//-------------------- HoneyBee SubClass
class HoneyBee extends Insect implements Pollinator, Builder
{
    private int pollinateRank;
    private int builderRank;

    HoneyBee(){
        //Default Constructor
    }

    HoneyBee(String type, String name, int pollinateRank, int builderRank)
    {
        setType(type);
        setName(name);
        this.pollinateRank = pollinateRank;
        this.builderRank = builderRank;
    }

    @Override
    public String purpose() //from abstract class 'Insect'
    {
        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!";
    }

    @Override
    public int getPollinateRank() //from interface "Pollinator"
    {
        return pollinateRank;
    }

    @Override
    public int getBuilderRank() //From interface "Builder"
    {
        return builderRank;
    }
}

//-------------------- Praying Mantis SubClass
```

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

    PrayingMantis(){
        //Default Constructor
    }

    PrayingMantis(String type, String name, int predatorRank)
    {
        setType(type);
        setName(name);
        this.predatorRank = predatorRank;
    }

    @Override
    public String purpose() //from abstract class "Insect"
    {
        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.";
    }

    @Override
    public int getPredatorRank() //from interface "Predator"
    {
        return predatorRank;
    }
}

//-------------------- Ant SubClass
class Ant extends Insect implements Builder, Decomposer, Predator
{
    private int builderRank;
    private int decomposeRank;
    private int predatorRank;

    Ant(){
        //Default Constructor
    }

    Ant(String type, String name, int builderRank, int predatorRank, int decomposeRank)
    {
        setType(type);
        setName(name);
        this.builderRank = builderRank;
        this.predatorRank = predatorRank;
        this.decomposeRank = decomposeRank;
    }

    @Override
    public String purpose() //from abstract class "Insect"
    {
        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!";
    }

```

```java
    @Override
    public int getBuilderRank() //from interface "Builder"
    {
        return builderRank;
    }

    @Override
    public int getDecomposeRank() //from interface "Decomposer"
    {
        return decomposeRank;
    }

    @Override
    public int getPredatorRank() //from interface "Predator"
    {
        return predatorRank;
    }
}

//--------------------LadyBug SubClass
class LadyBug extends Insect implements Pollinator, Predator
{
    private int pollinateRank;
    private int predatorRank;

    LadyBug(){
        //Default Constructor
    }

    LadyBug(String type, String name, int pollinateRank, int predatorRank)
    {
        setType(type);
        setName(name);
        this.pollinateRank = pollinateRank;
        this.predatorRank = predatorRank;
    }

    @Override
    public String purpose() //from abstract class "Insect"
    {
        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.";
    }

    @Override
    public int getPollinateRank() //from interface "Pollinator"
    {
        return pollinateRank;
    }

    @Override
    public int getPredatorRank() //from interface "Predator"
    {
        return predatorRank;
    }
}
```