1 **import** edu.sjcny.gpv1.\*;

2 **import** java.awt.\*;

3 **import** javax.swing.\*;

4

5 **public** **class** ArrayAlgorithms **extends** DrawableAdapter

6 {

7 **static** ArrayAlgorithms ge = **new** ArrayAlgorithms();

8 **static** GameBoard gb = **new** GameBoard(ge, "Array Algorithms");

9 **static** ParentSnowman[] parent;

10 **static** SnowChild[] child;

11

12 **public** **static** **void** main(String[] args)

13 {

14 String name;

15 parent = **new** ParentSnowman[5];

16 child = **new** SnowChild[5];

17

18 **for**(**int** i = 0; i < 5; i++)

19 {

20 name = JOptionPane.showInputDialog("enter a family name");

21 name = name.toUpperCase();

22 child[i] = **new** SnowChild(50 + 60, 80 + 90 \* i, name);

23 parent[i] = **new** ParentSnowman(50, 50 + 90 \* i, name);

24 }

25 showGameBoard(gb);

26 }

27

28 **public** **void** draw(Graphics g)

29 {

30 **for**(**int** i = 0; i<5; i++)

31 {

32 **if**(parent[i].getVisible() == true)

33 { parent[i].show(g);

34 child[i].show(g);

35 }

36 }

37 }

38

39 **public** **void** keyStruck(**char** key)

40 {

41 **int** index;

42

43 String sKey = Character.toString(key);

44 index = findValue(parent, sKey);

45 **if**(index != -1) **//name is valid, reverse family’s visibility**

46 {

47 **if**(parent[index].getVisible() == true)

48 {

49 parent[index].setVisible(false);

50 }

51 **else**

52 {

53 parent[index].setVisible(true);

54 }

55 }

56

57 **if** (key == 'U') **//up arrow struck, reverse visibility of min name**

58 {

59 index = findMin(parent); **//index of first family in alphabetic order**

60 **if**(parent[index].getVisible() == true)

61 {

62 parent[index].setVisible(false);

63 }

64 **else**

65 {

66 parent[index].setVisible(true);

67 }

68 }

69 **if**(key == 'S') **//sort the families**

70 {

71 selectionSort(parent);

72 }

73 }

74

75 **public** **static** **int** findValue(ParentSnowman[] parent, String targetValue)

76 {

77 **int** elementNumber = -1;

78 **for**(**int** i = 0; i< parent.length; i++)

79 {

80 **if**(parent[i].getName().equalsIgnoreCase(targetValue))

81 {

82 elementNumber = i;

83 **break**;

84 }

85 }

86 **return** elementNumber;

87 }

88

89 **public** **static** **int** findMin(ParentSnowman[] parent)

90 {

91 String min = parent[0].getName();

92 **int** elementNumber = 0;

93 **for**(**int** i = 1; i < parent.length; i++)

94 {

95 **if**(parent[i].getName().compareToIgnoreCase(min) < 0)

96 {

97 min = parent[i].getName();

98 elementNumber = i;

99 }

100 }

101 **return** elementNumber;

102 }

103

104 **public** **static** **void** selectionSort(ParentSnowman[] parent)

105 {

106 **int** iMin, tempInt;

107 ParentSnowman tempParent;

108 SnowChild tempChild;

109 String min;

110

111 **for** (**int** j = 0; j < parent.length; j++)

112 {

113 min = parent[j].getName();

114 iMin = j;

115 **for** (**int** i = j+1; i < parent.length; i++)

116 {

117 **if** (parent[i].getName().compareToIgnoreCase(min) < 0)

118 {

119 min = parent[i].getName();

120 iMin = i;

121 }

122 }

123 **if** ( iMin != j ) **//swap element j with minimum element**

124 {

125 tempParent = parent[j]; //swap array references

126 parent[j] = parent[iMin];

127 parent[iMin] = tempParent;

128 tempChild = child[j];

129 child[j] = child[iMin];

130 child[iMin] = tempChild;

131

132 tempInt = parent[j].getY(); **//swap Y positions**

133 parent[j].setY(parent[iMin].getY());

134 parent[iMin].setY(tempInt);

135 child[j].setY(parent[j].getY() + 30);

136 child[iMin].setY(parent[iMin].getY() + 30);

137 }

138 }

139 }

140 }

**Figure 6.24 The application ArrayAlgorithms.**