A shell program is built with the intention of evolving trees to be more fit.

Tree.java contains the DNA for what it means to be a Tree (graphic fractals in this case, but they look treeish). You only need to know about the existence of the following Tree methods:

**public Tree makeRandomTree()** //returns a random Tree fractal

**public Tree mutate(int mutationRate)** //given the current tree, return an almost-clone of the tree //where each field can vary by mutationRate

**public int compareTo(Object other)** //Trees compare to one another by how fit they are

//(one that is more fit is > one that is less fit)

In TreePanel.java, you need to know of the following data-fields:

**private static Tree [] forest;** //most fit is at forest[0]  
**private static int SAMPLE\_SIZE = 10;** //number of trees to mutate (2-25)   
**private static int MUTATION\_RATE = 5;** //percentage a tree can mutate (1-100)  
**private static int numGenerations;** //count the number of generations

A constructor for the Panel will populate the forest with random trees.

public TreePanel()  
 {  
 numGenerations = 0;  
 evolve = false;  
 forest = new Tree[SAMPLE\_SIZE];  
 forest[0] = forest[0].makeRandomTree();  
 for(int i=1; i<forest.length; i++)  
 forest[i] = forest[0];  
 }

Your job is detailed on the next page…

Code needs to be completed in TreePanel.java to do the following:

1. Complete the main brains of the AI to advance the evolution of trees by one generation.

private static void advanceGeneration()

{

numGenerations++;

//\*\*\*Complete the code here to find the most fit tree,

//\*\*\*place the most fit tree at index 0

//\*\*\*repopulate the rest of the forest with children of the most fit

//\*\*\*allowing each child to differ from the best by the MUTATION\_RATE

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

1. The client will have the option of pausing evolution (right mouse click) in order to adjust the sample size (# of children in forest) or the mutation rate. Complete the code to allow the client to make the forest one larger or one smaller. The more children there are, the more likely it will be that a child will be more fit than the parent, but will require more processing to find the most fit.

private void keyPressed(KeyEvent e)

{

if(e.getKeyCode()==KeyEvent.VK\_1) //Sample Size (2-25)  
 {  
 if(SAMPLE\_SIZE <= 2)  
 return;  
 SAMPLE\_SIZE--;   
 //\*\*\*COMPLETE THE CODE HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*make a new Tree array that is one-element smaller than forest

//\*\*\*copy elements from forest to the new smaller array

//\*\*\*change forest to be the new smaller array.

//\*\*\*Similar code should be done to increase the sample size by one.  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 repaint();  
 return;  
 }

if(e.getKeyCode()==KeyEvent.VK\_2) //Sample Size (2-25)  
 {  
 if(SAMPLE\_SIZE >= 25)  
 return;  
 SAMPLE\_SIZE++;  
 //\*\*\*COMPLETE THE CODE HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 repaint();  
 return;  
 }