A* Implementation

The algorithm proceeds as follows (with each step explained in further detail below):
1. Take in currNode as a parameter (initially the startNode)
2. Generate a list of valid adjacent Nodes
3. Process the adjacent Nodes and add them to the Open List
4. Take the lowest F-Value Node in the Open List and move it to the closed list
5. Repeat steps 1-4 until one of the adjacent Nodes is the destination Node

1.
The pathfinding method takes in a sinlge parameter, currNode, which is of type Node. Nodes have 5 fields, but only 4 of them are set explicitly. Each Node has a Point location, a parent Node, Heuristic Value, total Movement cost to that Node, and the sum of Heuristic and Movement costs. This last field is computed interally. Initially this is the start Node, with no parent Node.

Location- This is a simple java.awt.Point with X and Y values.

Parent Node- the Node that leads to this Node on the shortest path

Heuristic Value- Manhattan heuristic which estimates minimum cost to the destination Node as the adjacent and oposite lengths on a right triangle from this Node to the destination Node. I chose this heuristic because it is easy to precompute and store in a 2D array.

Movement Cost- I use a cost of 10 for horrizontal and veritcal movements, and a cost of 14 for diagonal movements. If you have a triangle with opposte and adjacent lengths of 1, the hypotenuse is sqrt(2)~=1.4 per the pythagorean theorem. I multiplied by 10 to keep as an integer while gaining precision. The next digit in sqrt(2) is 1 so further precision is hardly needed. Total Movement cost will be covered when discussing the algorithm.

F-Value- The sum of the Heuristic (H-value) and movement cost (G-value)

2.
From the current Node we want to find all valid moves, and rank them in some way. First, look at the 8 Nodes touching or diagonal to the currNode. If the ping(Point) method returns that the spot is a valid move then create a new Node and initialize it with the following values: Heuristic Value= value looked up in precomputed list, parent Node = currNode, MovementCost = parent movement cost + horrizontal or diagonal movement cost as appropriate. The F-Value will be computed internally once the Node is initialized. We collect all the valid moves and return them back to the pathfinding method.

3.
For each Node in the adjacent Node list there are four cases:

Case 1: The node is the destination Node, return it. We are done
Case 2: The node is in the ClosedList, move on to the next Node
Case 3: The Node is in the Open List, reparent the Node in OpenList if the Node's Movement cost + currNode Movement Cost < the Node's FValue
Case 4: We have never seen this Node before, add it to the open List

4.
I use a Priority Queue to store the Nodes in the Open List. This makes for O(1) lookup of the Node with the Lowest F-Value. The tradeoff being that insertion is slower than a List, but we are willing to take this hit because as the OpenList grows in size the constant lookup of lowest F-Value will payoff greatly.

5. Pop the Lowest F-Value Node off the OpenList and move it to the ClosedList, repeat the algorithm with the popped Node as the new CurrNode

Performance

I tested the algorithm on the 4 given test cases plus two others (one 40x40 world and one 10x10 world with arrow shaped walls).

On testInput1: 3 moves, 21 pings
On testInput2: 7 moves, 100 pings
On testInput3: 13 moves, 599 pings
On testInput4: 38 moves, 1758 pings
On customInput1: 65 moves, 8913 pings
On customInput2: 19 moves, 399 pings

On customInput2 it was interesting to note that the openList will fill up the arrow or L shaped wall before finding the way around the arrow or L shaped wall, I believe this is the worst performance possible from the algorithm.

Uncertainty

To handle pings that can return the wrong value the further the robot is from the spot, I move the robot to the Node that was just popped off the queue before calling the method that finds adjacent Nodes. After pings are done the robot moves back to the start. This dramatically increases moves by the size of the Closed List while leaving the number of pings unchanged.