```java
import java.awt.*;
import java.util.*;

/**
   Aggressive pack animal attracted to the smell of blood. Wolves
   share information about targets that they have seen and congregate
   on victims.
   <p>

   <p>
   Sound effects from http://www.ualberta.ca/~jzgurski/wcomm.html

   <p>Morgan McGuire
   <br>morgan@cs.williams.edu


<h3>State Machine</h3>
<pre>

.-----------EXPLORE
|    Timeout   ^\
| and loss of /  \
|   target   /    \ Any Wolf
|   after a /      \observes
|    kill   \      /a Target
|            \    v
|           PURSUE
|   Any Wolf ^   \
|   observes/     \Reach location
|   a target\     /or Timeout
|            \   v
'------------>HUNT
No unexplored
 squares left
</pre>
*/
public class WolfMM extends Creature {

    /** If true, hides the toString method */
    final static private boolean RELEASE_MODE = true;

    final protected Random random = new Random();

    private enum State {
        /** Seeking to cover the map */
        EXPLORE,

        /** Chasing a recently observed target */
        PURSUE,

        /** Seeking targets */
        HUNT,
    };


/////////////////////////////////////////////////////////////////////////
    /** Allocated by the first creature to spawn. */
    static private Map         map;
    static private int         numWolves         = 0;
    static private int         initialNumWolves  = 0;
    static private boolean     firstRound        = true;
    static private long        lastKillTime      = 0;

    /** If this classId is ever set, it is treated as nonzero */
    static private int         friendClassId = -1;


    /** A kill is recent if it has occured within this much game time
     * of now */
    static private final long RECENT_KILL =
            Simulator.MOVE_FORWARD_COST * 20;

    /** Number of turns that a creature will hunt before it changes
        target, unless it encounters an enemy in the mean time. */
    static private final int MAX_HUNT_TURNS = 20;

    /** Number of turns that a creature will pursue a target before it
        checks to see if there is a closer target.  If this is too
        large, then the creature will run right past near targets.  If
        it is too small, then the creature may become indecisive and
        keep switching targets if pathfinding requires it to walk a
        larger distance.`*/
    static private final int MAX_PURSUIT_TURNS = 14;


////////////////////////////////////////////////////////////////////////

    /** State-machine state of this Wolf */
    private State              state             = State.EXPLORE;

    /** Where this creature is currently trying to go */
    private Point              target            = null;

    /** How many turns (actions) since this Wolf entered the
      current state;
      used for timeout transitions. */
    private int                turnsInState      = 0;

    /** Last thought, for debugging */
    private String             thinking          = "";
```

```java
    /** True if there is some non-hostile object blocking this square */
    private boolean nonHostileBlock(Observation obs) {
        return (obs != null) &&
            ((obs.classId == myClassId()) ||
             (obs.classId == WALL_CLASS_ID) ||
             (obs.classId == HAZARD_CLASS_ID));
    }


    /** Version of isEnemy that returns false for null */
    @Override
    protected boolean isEnemy(Observation obs) {
        return (obs != null) && (obs.classId != friendClassId) &&
super.isEnemy(obs);
    }


    /** If there is some situation that demands a specific action
     independent of the state, resolve it and return true, otherwise
     return false.*/
    private boolean overridingSituation() {
        final Point p = getPosition();

        Observation N = map.get(p.x, p.y - 1);
        Observation S = map.get(p.x, p.y + 1);
        Observation E = map.get(p.x + 1, p.y);
        Observation W = map.get(p.x - 1, p.y);

        // Am I unable to move?
        if (nonHostileBlock(N) && nonHostileBlock(S) &&
            nonHostileBlock(E) && nonHostileBlock(W)) {
            think("Overriding situation: I can't move");
            delay();
            return true;
        }

        // Is there a known enemy right next to me?
        if (isEnemy(map.get(getDirection().forward(p)))) {
            think("Overriding situation: Enemy in front");
            // Prefer to attack forward
            attack();
            return true;
        } else if (isEnemy(N)) {
            think("Overriding situation: Enemy to the North");
            turn(Direction.NORTH);
            attack();
            return true;
        } else if (isEnemy(S)) {
            think("Overriding situation: Enemy to the South");
            turn(Direction.SOUTH);
            attack();
            return true;
        } else if (isEnemy(E)) {
            think("Overriding situation: Enemy to the East");
            turn(Direction.EAST);
            attack();
            return true;
        } else if (isEnemy(W)) {
            think("Overriding situation: Enemy to the West");
            turn(Direction.WEST);
            attack();
            return true;
        }

        return false;
    }


    …
}
```