```java
import java.util.ArrayList;
import java.util.List;

class Pattern{
    /**
     * This class contains the pattern of data that is given out over each second
     * The Pattern starts at the beginning of a second. This means that for blocks that
     * dont begin at the start of a second, we need to track the offset.
     */

     private List<TickPattern> pattern;
     private int finalPatternSize = -1; // I got into the habit of setting initial values to stuff I know shouldn't exist
     private int index = -1;

     // In order to correctly order the data from multiple blocks into human readable forms like graphs, the time of each
     // data point should be stored. To do this I will save the nearest time to the block start time and the number of seconds
     // That has elapsed since. Then from the tick number of the pattern one can calculate the ms time since the logger started that the
     // data point accured.
     // If one knows the global time at which the logger started then data from lots of loggers can be combined with many different channels
     // and many different blocks to give a human readable graph.
     private long nearestSecondToBlockStartTime = 0;
     private int numberOfSecondsElapsedFromStartTime = 0;

     public Pattern(){
         pattern = new ArrayList<>();
     }

     public List<TickPattern> getPattern(){
         return pattern;
     }

     public void add(TickPattern tickPattern){
         pattern.add(tickPattern);
     }

     /**
      * This function works out the starting index of the pattern in the block
      */
     public void setStartIndex(long blockStartTime){
         // number of milli seconds into second that the block began
        int offset = blockStartTime % 1000;
        nearestSecondToBlockStartTime = blockStartTime - (long)offset;
        //Calculate the number of positions left in the second before the pattern recurs
        // This is the number milliseconds left in the first non-complete second
        this.index = findIndexOfNearestTickToOffset(offset);
     }

     /**
      * Function to find the nearest tick to the offset provided by the block start time.
      */
    private int findIndexOfNearestTickToOffset(int offset){
     // pattern size should be fixed at this point
     this.finalPatternSize = pattern.size();
        for(int k=0; k < finalPatternSize ; k++ ){
            TickPattern tickPattern = pattern.get(k);
            if(tickPattern.getTick() >= offset){
                // Return the index as soon as the tick in the pattern is greater than
                //or equal to the offset provided by the block start time.
                return k;
            }
        }
        // If this function has not returned a value by this point it means that the offset in the second
        // was higher than any of the ticks in the pattern. If this is the case than
```

```java
            the next data point
            // will be from the next second at tick t=0.
            return 0;
        }

        public incrementIndex(){
            this.index++;
            if(this.index == finalPatternSize){
                this.index = 0;
                //every time it ticks over add a full second on to counter above
                this.numberOfSecondsElapsedFromStartTime++;
            }
        }

        public int getChannelIdForCurrentIndex(){
            TickPattern tickPattern = pattern.get(index);
            return tickPattern.getChannelId();
        }

        /**
         * The tick value is the millisecond value that the logger is fired on
         therefore knowing the block start time
         * and the number of secs elasped with the tick value you can calculate raw time
         */
        private long calculateTimeOfDataPoint(int tickValue){
           return nearestSecondToBlockStartTime + (numberOfSecondsElapsedFromStartTime *
            1000) + tickValue;
        }

        public ChannelTime getInfoForCurrentIndex(){
            TickPattern tickPattern = pattern.get(index);
            return new ChannelTime(tickPattern.getChannelId(), calculateTimeOfDataPoint(
            tickPattern.getTick()));
        }

        public DataPoint createDataPointFromPattern(int value){
            TickPattern tickPattern = pattern.get(index);
            DataPoint dataPoint = new DataPoint(value, time)
        }

        /*
        public List<TickPattern> getPatternForTick(int tick){
            // todo if needed
            return null;
        }
        */
    }
```