```java
import java.util.Random;

public class ProofOfConcept {

    private Map1L<String, Map1L<Integer, Double>> teamData;

    private void createNewRep() {
        this.teamData = new Map1L<String, Map1L<Integer,
Double>>();
    }

    /**
     * No-argument constructor.
     */
    public ProofOfConcept() {
        this.createNewRep();
    }

    // Kernel Methods

    public void addCustomStatistic(String category, int rank,
double value) {

        if (this.teamData.hasKey(category)) {
            Map1L<Integer, Double> newStatistic = new
Map1L<>();
            newStatistic.add(rank, value);
            this.teamData.replaceValue(category, newStatistic);
        } else {
            Map1L<Integer, Double> newCategoryData = new
Map1L<>();
            newCategoryData.add(rank, value);
            this.teamData.add(category, newCategoryData);
        }
    }

    public void removeStatistic(String category) {
        SimpleWriter out = new SimpleWriter1L();
        if (this.teamData.hasKey(category)) {
            this.teamData.remove(category);
        } else {
            out.println("Statistic Not Found");
        }
        out.close();
```

```
46      }
47
48      public Map1L<Integer, Double>
    getStatisticsByCategory(String category) {
49  //          returns a map with the key being the statistics rank,
    and the value of the map being the value of the value of the
    statistic
50          return this.teamData.value(category);
51
52      }
53
54      public Sequence1L<String> getAllCategories() {
55  //          return a list of the all the keys in this
56
57          Sequence1L<String> categories = new Sequence1L<>();
58
59          for (Map1L.Pair<String, Map1L<Integer, Double>> pair :
    this.teamData) {
60              categories.add(0, pair.key());
61          }
62
63          return categories;
64      }
65
66      public void bestAndWorstStatistics() {
67          SimpleWriter out = new SimpleWriter1L();
68          Map1L<String, Integer> bestStatistics = new Map1L<>();
69          Map1L<String, Integer> worstStatistics = new Map1L<>();
70
71          // Iterate over the teamData map to find the best and
    worst statistics
72          for (Map1L.Pair<String, Map1L<Integer, Double>> team :
    this.teamData) {
73              String category = team.key();
74              Map1L<Integer, Double> teamValues = team.value();
75              for (Map1L.Pair<Integer, Double> teamPair :
    teamValues) {
76                  int rank = teamPair.key();
77
78                  // Logic for best statistics
79                  if (bestStatistics.size() < 5) {
80                      bestStatistics.add(category, rank);
81                  } else {
82                      int worstRank = Integer.MAX_VALUE;
```

```
 83                     String worstCategory = null;
 84                     for (Map1L.Pair<String, Integer>
    bestStats : bestStatistics) {
 85                         int currentRank = bestStats.value();
 86                         if (currentRank < worstRank) {
 87                             worstRank = currentRank;
 88                             worstCategory = bestStats.key();
 89                         }
 90                     }
 91                     if (rank > worstRank) {
 92                         bestStatistics.remove(worstCategory);
 93                         bestStatistics.add(category, rank);
 94                     }
 95                 }
 96 
 97                 // Logic for worst statistics
 98                 if (worstStatistics.size() < 5) {
 99                     worstStatistics.add(category, rank);
100                 } else {
101                     int bestRank = -1; // Initialize to minimum
    possible value
102                     String bestCategory = null;
103                     for (Map1L.Pair<String, Integer>
    worstStats : worstStatistics) {
104                         int currentRank = worstStats.value();
105                         if (currentRank > bestRank) {
106                             bestRank = currentRank;
107                             bestCategory = worstStats.key();
108                         }
109                     }
110                     if (rank < bestRank) {
111                         worstStatistics.remove(bestCategory);
112                         worstStatistics.add(category, rank);
113                     }
114                 }
115             }
116         }
117 
118         // Print out the best statistics names
119         out.println("Top 5 statistics:");
120         for (Map1L.Pair<String, Integer> bestStat :
    bestStatistics) {
121             out.println(bestStat.key());
122         }
```

```
123
124         // Print out the worst statistics names
125         out.println("Bottom 5 statistics:");
126         for (Map1L.Pair<String, Integer> worstStat :
    worstStatistics) {
127             out.println(worstStat.key());
128         }
129
130         out.close();
131     }
132
133     // Secondary Methods
134
135     public static void runSimulation(ProofOfConcept team1,
136             ProofOfConcept team2) {
137
138         // using the stats from each team and all the
139         // methods above, come up with a simple
140         // method to try and predict the
141         // outcome of the game, and also use some sort of
142         // randomness so each simulation
143         // isn't the same
144
145         // for now ill just generate a random number between -2
    and 1.
146         // if the number is positive, team1 wins, otherwise
    team2 wins.
147         // reason i'm using -2 to 1 instead of -1 to 1 is
    because team2
148         // typically wins more often because they have home
    court advantage.
149
150         //* good to note that team1 should represent the away
    team
151         // and team2 represents the home team *
152
153         Random random = new Random();
154         double randomValue = -2 + (1 - (-2)) *
    random.nextDouble();
155
156         if (randomValue >= 0) {
157             System.out.println("Team 1 wins!");
158         } else {
159             System.out.println("Team 2 wins!");
```

```
160         }
161     }
162 
163     public static void main(String[] args) {
164         ProofOfConcept ohioState = new ProofOfConcept();
165         ProofOfConcept nebraska = new ProofOfConcept();
166 
167         // Adding statistics for Ohio State
168         ohioState.addCustomStatistic("3PT Percentage", 25,
    0.37);
169         ohioState.addCustomStatistic("Free Throw Percentage",
    15, 0.82);
170         ohioState.addCustomStatistic("Rebounds per Game", 35,
    42);
171         ohioState.addCustomStatistic("Assists per Game", 18,
    22);
172         ohioState.addCustomStatistic("Steals per Game", 5, 8);
173         ohioState.addCustomStatistic("Blocks per Game", 2, 4);
174         ohioState.addCustomStatistic("Field Goal Percentage",
    40, 0.48);
175         ohioState.addCustomStatistic("Turnovers per Game", 10,
    14);
176         ohioState.addCustomStatistic("Points per Game", 60,
    80);
177         ohioState.addCustomStatistic("Defensive Rating", 60,
    75);
178 
179         // Adding statistics for Nebraska
180         nebraska.addCustomStatistic("3PT Percentage", 20,
    0.34);
181         nebraska.addCustomStatistic("Free Throw Percentage",
    12, 0.75);
182         nebraska.addCustomStatistic("Rebounds per Game", 32,
    38);
183         nebraska.addCustomStatistic("Assists per Game", 14,
    18);
184         nebraska.addCustomStatistic("Steals per Game", 4, 7);
185         nebraska.addCustomStatistic("Blocks per Game", 1, 3);
186         nebraska.addCustomStatistic("Field Goal Percentage",
    35, 0.45);
187         nebraska.addCustomStatistic("Turnovers per Game", 12,
    16);
188         nebraska.addCustomStatistic("Points per Game", 55, 75);
189         nebraska.addCustomStatistic("Defensive Rating", 65,
```

```
    80);
190
191         // Ideally the user wouldnt manually enter the stats
    like this.
192         // They would automatically by scraping data or pulling
    from
193         // a database or something.
194
195         ohioState.bestAndWorstStatistics(); // can be used to
    compare the strengths and weakness of each team
196         nebraska.bestAndWorstStatistics();
197
198         runSimulation(ohioState, nebraska);
199     }
200 }
201
```