```java
import components.map.Map1L;

public class CollegeBasketballTeam1
        extends CollegeBasketballTeamComponentSecondary {

    /**
     * Kernel implementation of CollegeBasketballTeam using a Map of Maps
     * representation.
     *
     * Convention: The outer Map stores statistics categorized by StatCategory.
     * Each StatCategory is mapped to an inner Map, which stores statistics
     * ranked by an integer value. The statistics are stored as (rank, value)
     * pairs.
     *
     * Correspondence: Each StatCategory maps to an inner Map containing the
     * ranked statistics for that category. The outer Map represents the entire
     * collection of statistics for the team.
     */

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

    /**
     * Creator of initial representation.
     */
    private void createNewRep() {
        this.teamData = new Map1L<>();
    }

    /**
     * Default constructor.
     */
    public CollegeBasketballTeam1() {
        this.createNewRep();
    }

    /**
     * Adds a custom statistic to the team's data.
```

```
41      *
42      * @param category
43      *            the category of the statistic
44      * @param rank
45      *            the rank of the statistic
46      * @param value
47      *            the value of the statistic
48      * @requires category != null && value != null
49      * @ensures the custom statistic is added to the team's
    data
50      */
51     @Override
52     public void addCustomStatistic(StatCategory category, int
    rank,
53             double value) {
54         if (this.teamData.hasKey(category)) {
55             Map1L<Integer, Double> newStatistic = new
    Map1L<>();
56             newStatistic.add(rank, value);
57             this.teamData.replaceValue(category, newStatistic);
58         } else {
59             Map1L<Integer, Double> newCategoryData = new
    Map1L<>();
60             newCategoryData.add(rank, value);
61             this.teamData.add(category, newCategoryData);
62         }
63     }
64
65     /**
66      * Removes a statistic category from the team's data.
67      *
68      * @param category
69      *            the category of the statistic to remove
70      * @requires category != null
71      * @ensures the specified statistic category is removed
    from the team's data
72      */
73     @Override
74     public void removeStatistic(StatCategory category) {
75         SimpleWriter out = new SimpleWriter1L();
76         if (this.teamData.hasKey(category)) {
77             this.teamData.remove(category);
78         } else {
79             out.println("Statistic Not Found");
```

```
 80         }
 81         out.close();
 82     }
 83 
 84     /**
 85      * Retrieves statistics by category for the team.
 86      *
 87      * @param category
 88      *            the category of statistics to retrieve
 89      * @return a map containing the statistics for the
    specified category
 90      * @requires category != null
 91      * @ensures returns a map containing the statistics for the
    specified
 92      *          category
 93      */
 94     @Override
 95     public Map1L<Integer, Double> getStatisticsByCategory(
 96             StatCategory category) {
 97 
 98         return this.teamData.value(category);
 99     }
100 
101     /**
102      * Retrieves all categories of statistics for the team.
103      *
104      * @return a sequence containing all statistic categories
105      * @ensures returns a sequence containing all statistic
    categories
106      */
107     @Override
108     public Sequence1L<StatCategory> getAllCategories() {
109 
110         Sequence1L<StatCategory> categories = new
    Sequence1L<>();
111 
112         for (Map1L.Pair<StatCategory, Map1L<Integer, Double>>
    pair : this.teamData) {
113             categories.add(0, pair.key());
114         }
115 
116         return categories;
117     }
118 
```

```
119     /**
120      * Identifies the top 5 and bottom 5 statistics for the
    team.
121      *
122      * @ensures the top 5 and bottom 5 statistics for the team
    are identified
123      */
124     @Override
125     public void bestAndWorstStatistics() {
126 
127         SimpleWriter out = new SimpleWriter1L();
128         Map1L<StatCategory, Integer> bestStatistics = new
    Map1L<>();
129         Map1L<StatCategory, Integer> worstStatistics = new
    Map1L<>();
130 
131         // Iterate over the teamData map to find the best and
    worst statistics
132         for (Map1L.Pair<StatCategory, Map1L<Integer, Double>>
    team : this.teamData) {
133             StatCategory category = team.key();
134             Map1L<Integer, Double> teamValues = team.value();
135             for (Map1L.Pair<Integer, Double> teamPair :
    teamValues) {
136                 int rank = teamPair.key();
137 
138                 // Logic for best statistics
139                 if (bestStatistics.size() < 5) {
140                     bestStatistics.add(category, rank);
141                 } else {
142                     int worstRank = Integer.MAX_VALUE;
143                     StatCategory worstCategory = null;
144                     for (Map1L.Pair<StatCategory, Integer>
    bestStats : bestStatistics) {
145                         int currentRank = bestStats.value();
146                         if (currentRank < worstRank) {
147                             worstRank = currentRank;
148                             worstCategory = bestStats.key();
149                         }
150                     }
151                     if (rank > worstRank) {
152                         bestStatistics.remove(worstCategory);
153                         bestStatistics.add(category, rank);
154                     }
```

```
155                 }
156
157                 // Logic for worst statistics
158                 if (worstStatistics.size() < 5) {
159                     worstStatistics.add(category, rank);
160                 } else {
161                     int bestRank = -1; // Initialize to minimum
    possible value
162                     StatCategory bestCategory = null;
163                     for (Map1L.Pair<StatCategory, Integer>
    worstStats : worstStatistics) {
164                         int currentRank = worstStats.value();
165                         if (currentRank > bestRank) {
166                             bestRank = currentRank;
167                             bestCategory = worstStats.key();
168                         }
169                     }
170                     if (rank < bestRank) {
171                         worstStatistics.remove(bestCategory);
172                         worstStatistics.add(category, rank);
173                     }
174                 }
175             }
176         }
177
178         // Print out the best statistics names
179         out.println("Top 5 statistics:");
180         for (Map1L.Pair<StatCategory, Integer> bestStat :
    bestStatistics) {
181             out.println(bestStat.key());
182         }
183
184         // Print out the worst statistics names
185         out.println("Bottom 5 statistics:");
186         for (Map1L.Pair<StatCategory, Integer> worstStat :
    worstStatistics) {
187             out.println(worstStat.key());
188         }
189
190         out.close();
191
192     }
193
194     @Override
```

```java
    public void transferFrom(CollegeBasketballTeam team2) {

        // Clear the current team's data
        this.clear();

        // Iterate over the categories of team2 and transfer their statistics
        Sequence1L<StatCategory> categories = team2.getAllCategories();
        for (int i = 0; i < categories.length(); i++) {
            StatCategory category = categories.entry(i);
            Map1L<Integer, Double> stats = team2
                    .getStatisticsByCategory(category);

            // Transfer each statistic from team2 to this team
            for (Map1L.Pair<Integer, Double> pair : stats) {
                int rank = pair.key();
                double value = pair.value();
                this.addCustomStatistic(category, rank, value);
            }
        }

        // Clear team2's data after transfer
        team2.clear();
    }

    @Override
    public void clear() {
        this.teamData.clear();
    }

    @Override
    public CollegeBasketballTeam1 newInstance() {
        return new CollegeBasketballTeam1();
    }
}

```