THE PROBLEM?

When you did all the work in a group project

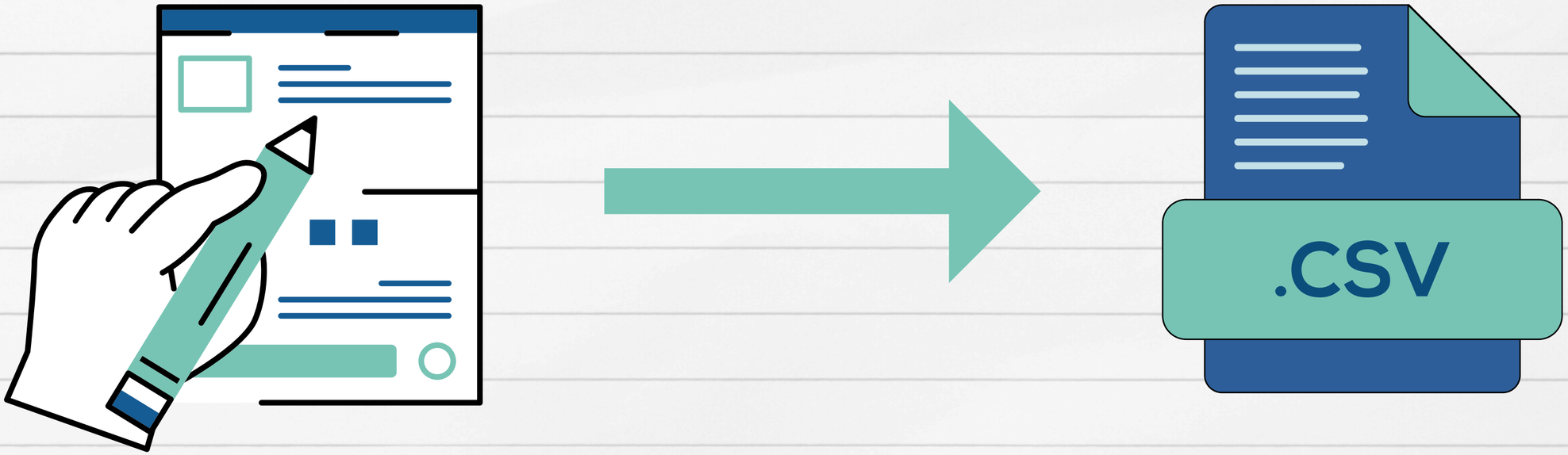
THE ULTIMATE TEAM MAKER

HOW DID WE IMPLEMENT IT?

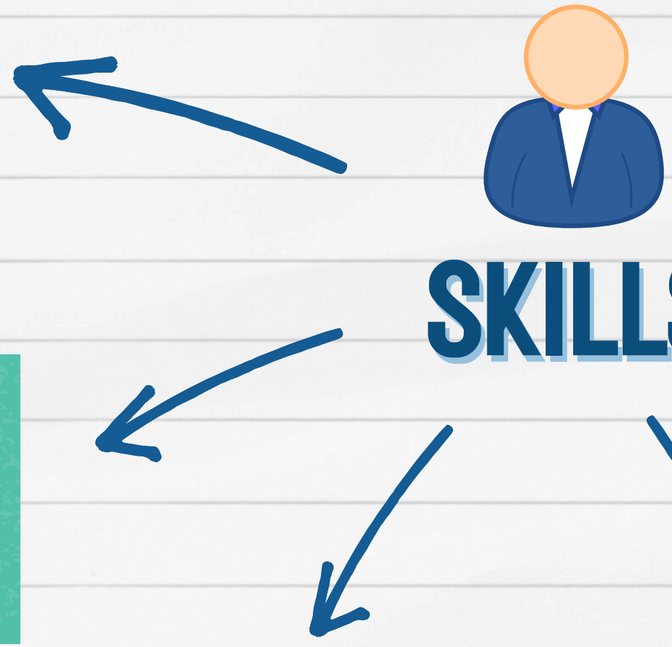
-THE ALGORITHM-

CONCEPT

COLLECTING USER DATA



CATEGORIZING SKILLS



TECHNICAL SKILLS

ANALYTICAL SKILLS

PHYSICAL SKILLS

CREATIVE SKILLS

SOFT SKILLS

PRESENTATION SKILLS

ASSIGNING WEIGHTS

TECHNICAL SKILLS

ANALYTICAL SKILLS

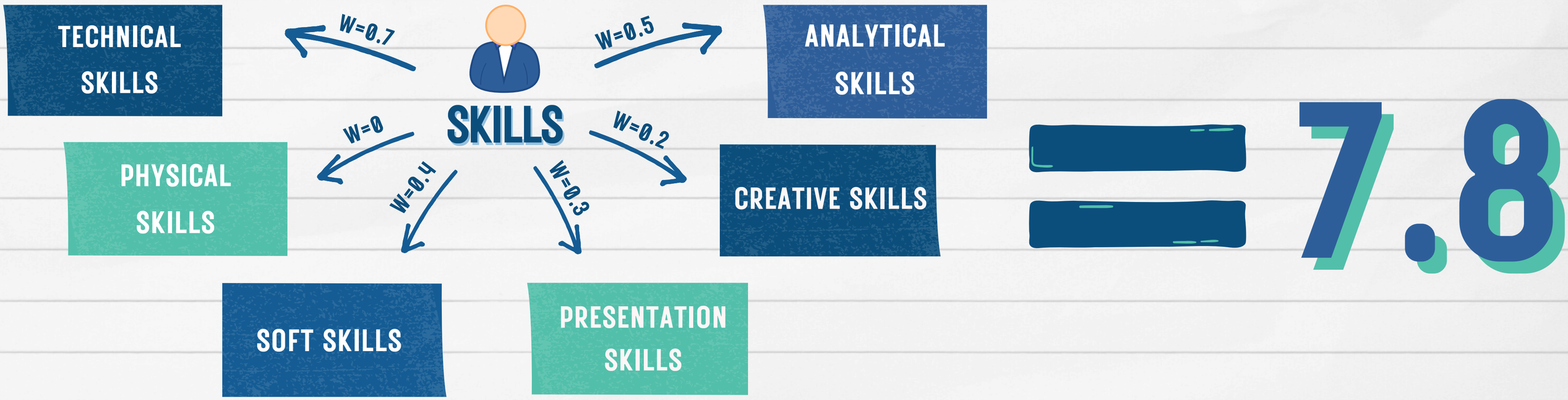
PHYSICAL SKILLS

CREATIVE SKILLS

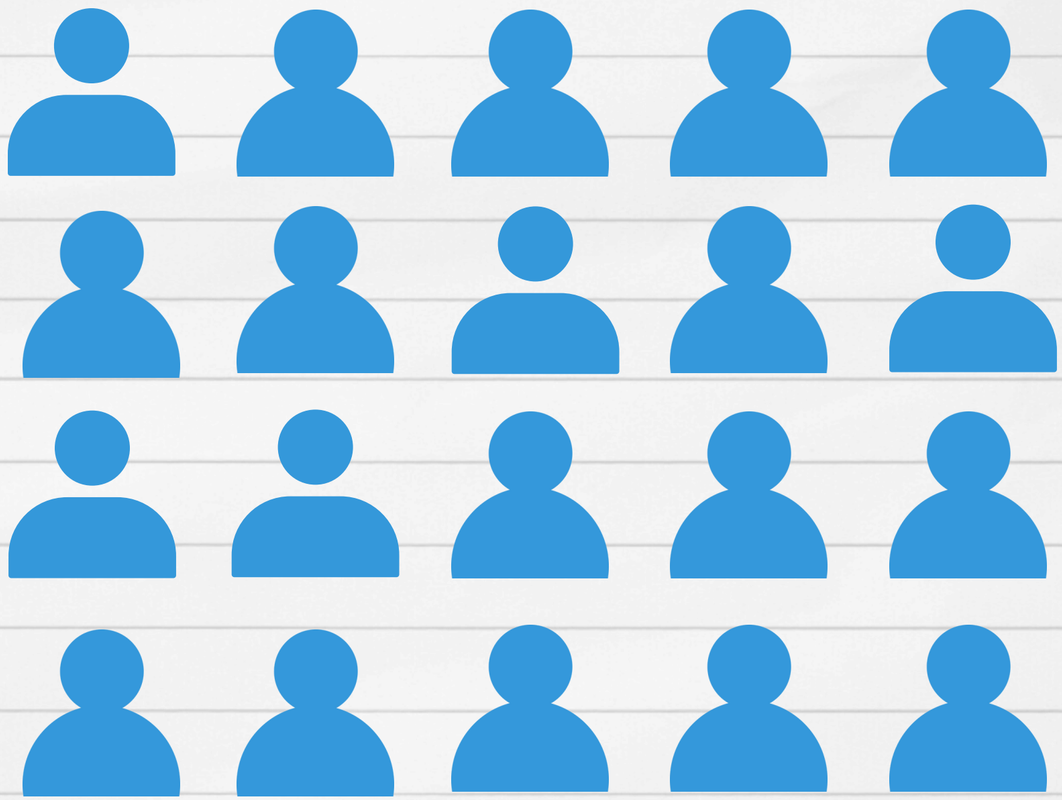
SOFT SKILLS

PRESENTATION SKILLS

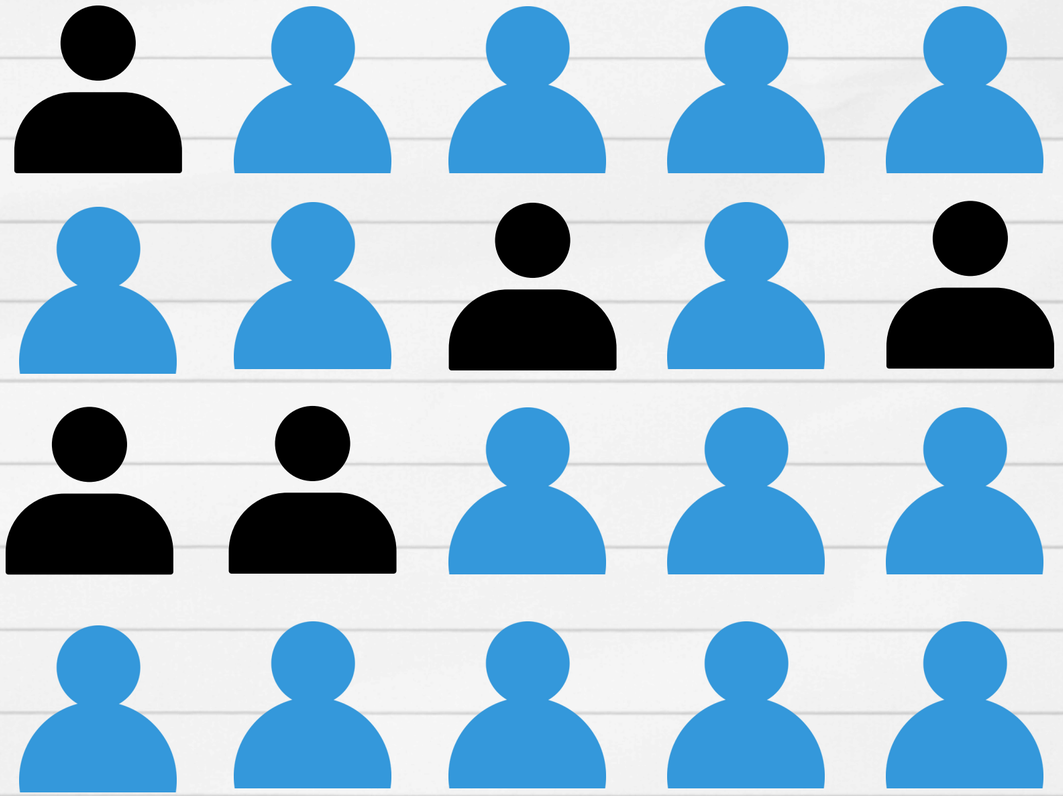
FINAL SCORE


TECHNICAL SKILLS
W=0.7
W=0.5
ANALYTICAL SKILLS
SKILLS
W=0
W=0.2
PHYSICAL SKILLS
W=0.4
W=0.3
CREATIVE SKILLS
SOFT SKILLS
PRESENTATION SKILLS
= 7.8

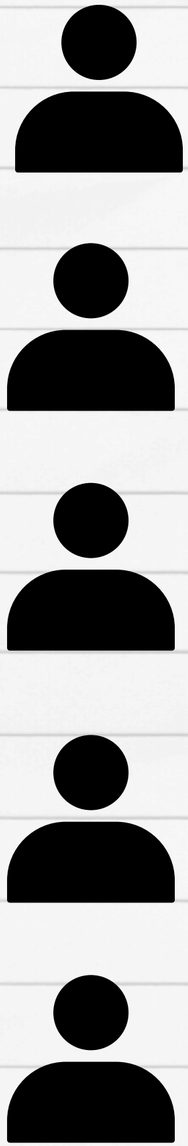
RANDOMTEAMGENERATOR

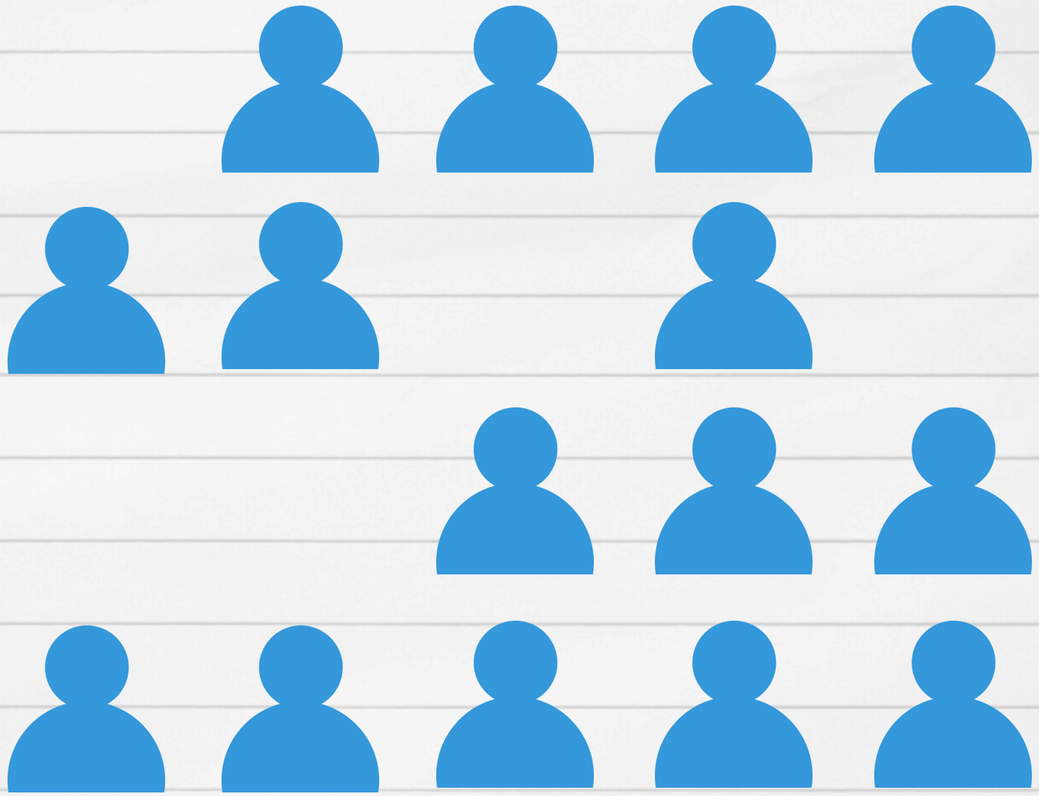


RANDOMTEAMGENERATOR

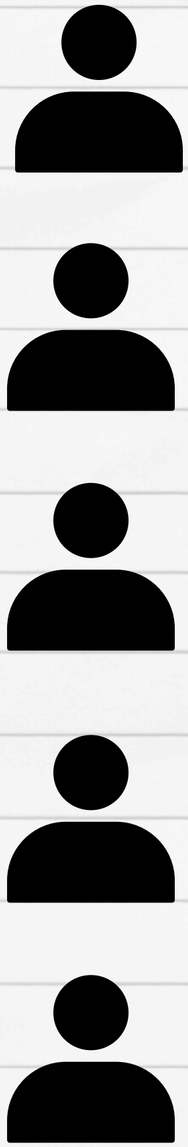


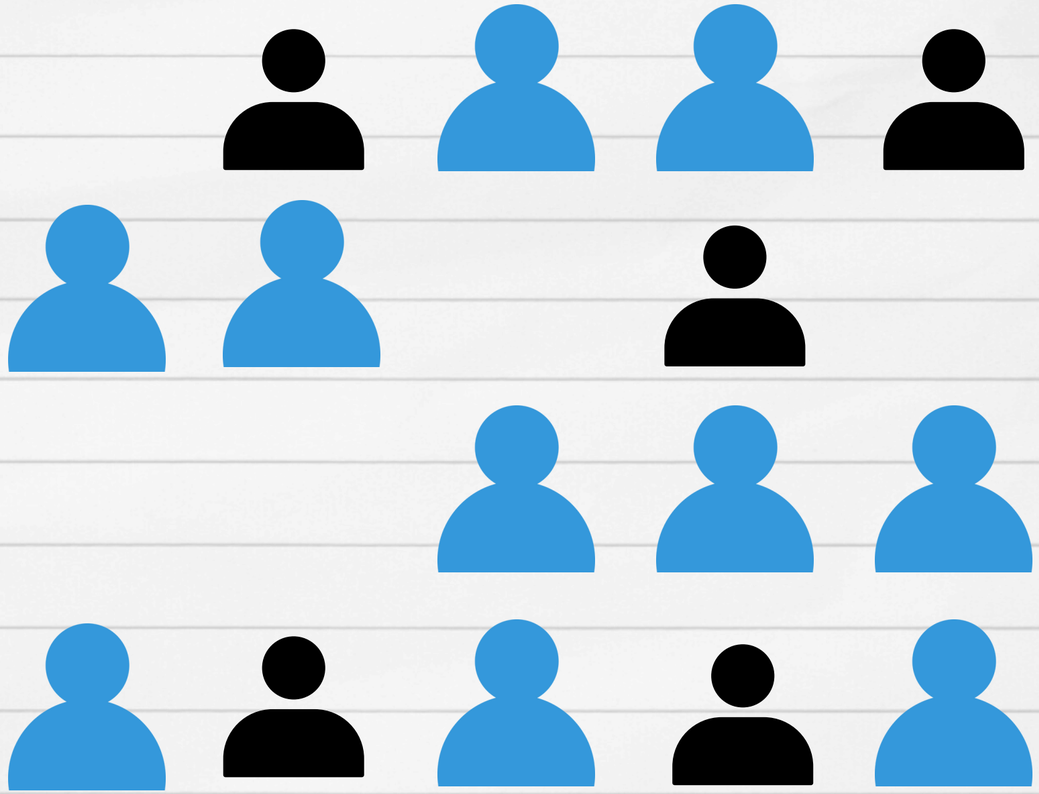
RANDOMTEAMGENERATOR



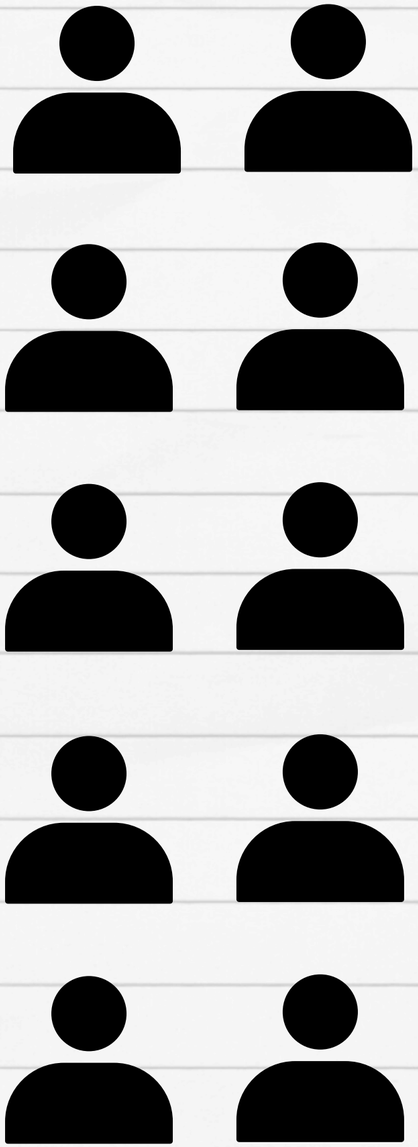


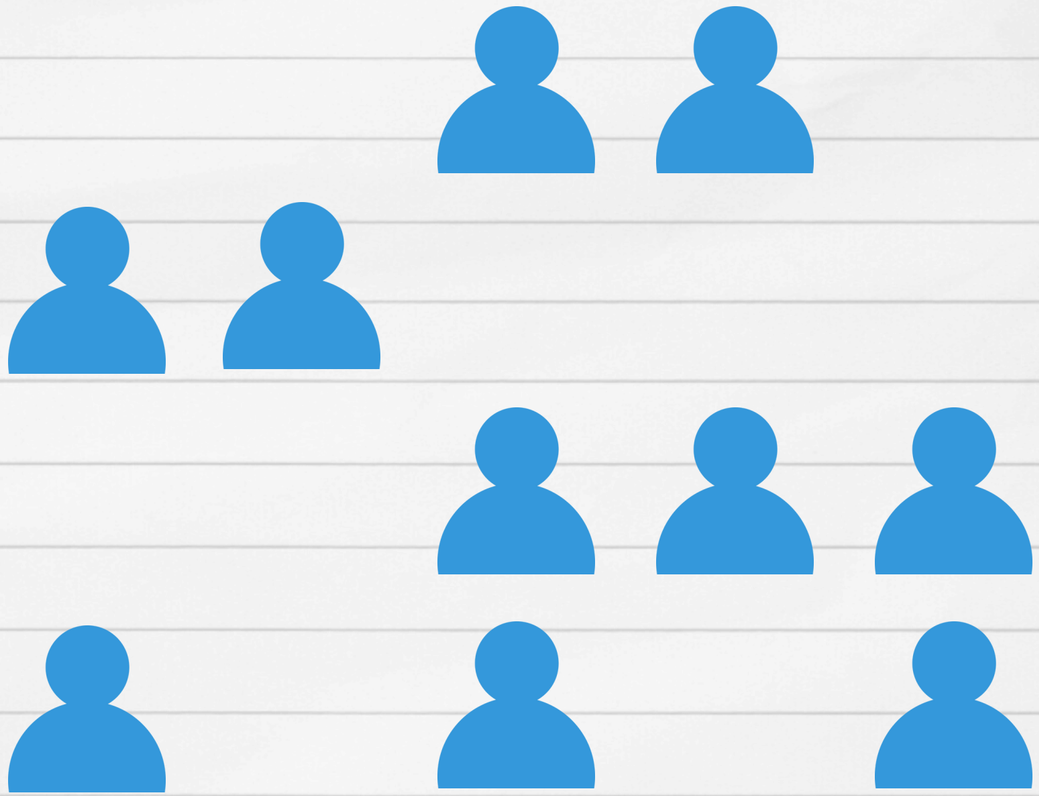
RANDOMTEAMGENERATOR

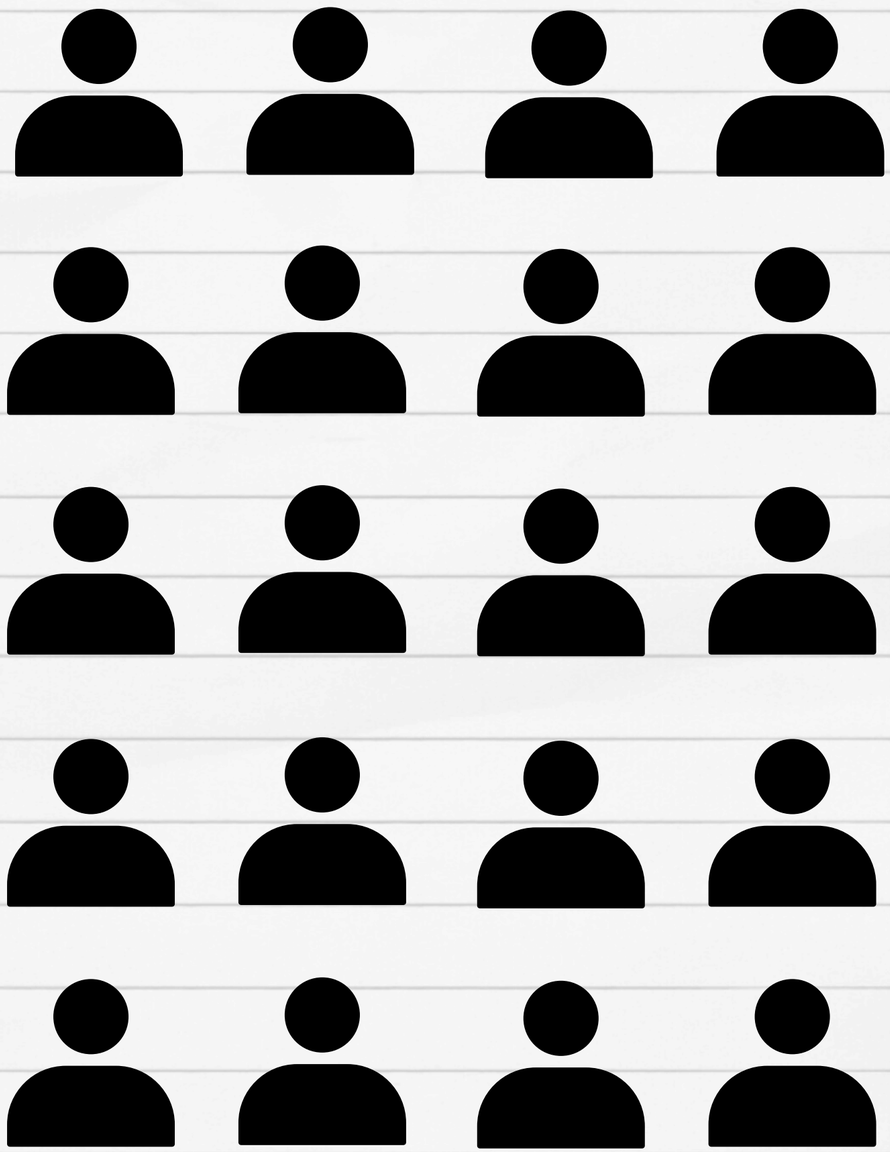




RANDOMTEAMGENERATOR







RANDOMTEAMGENERATOR

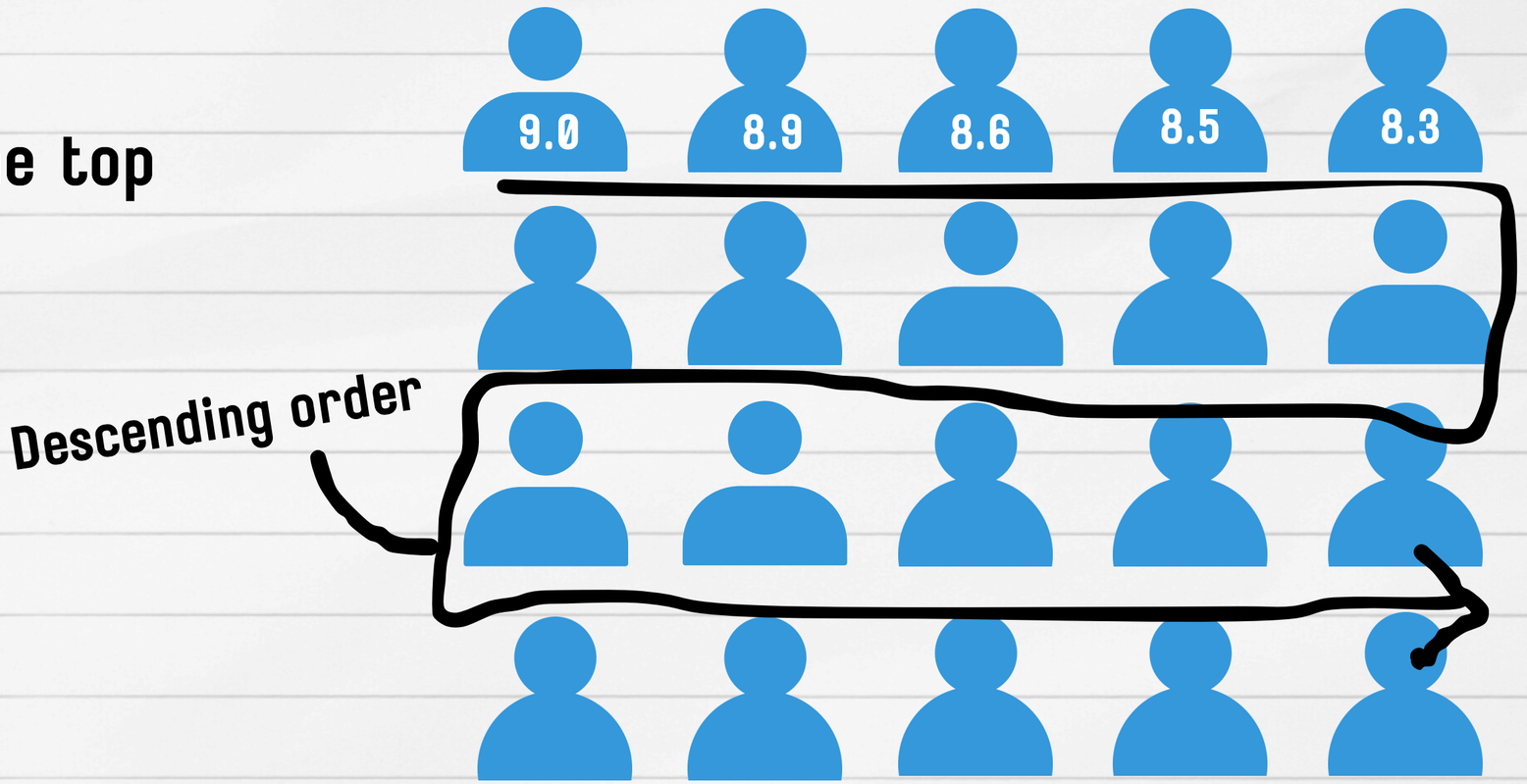
```
vector<Team> RandomTeamGenerator::createRandomTeams(int numTeams) {
    random_device rd;
    mt19937 gen(rd());

    vector<Team> teams(numTeams);
    int PersonIndex = 0;
    for (const auto& Person : Persons) {
        int teamIndex = PersonIndex % numTeams;
        teams[teamIndex].addPerson(Person);
        PersonIndex++;
    }

    return teams;
}
```

-TEST CASE-

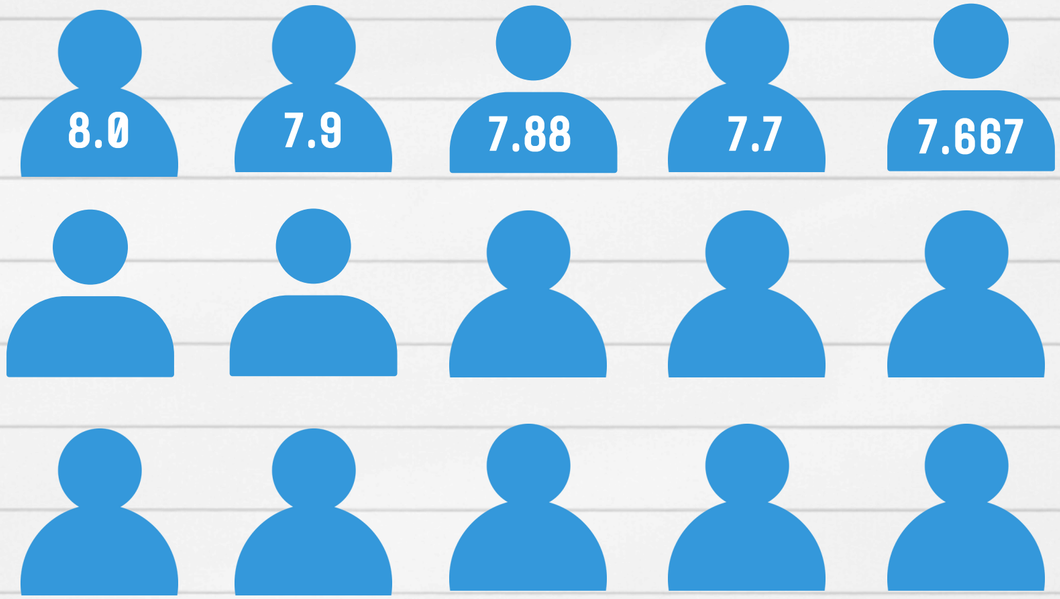
RANDOM CATEGORICALTEAMGENERATOR

1) Arrange the students in a list in descending order

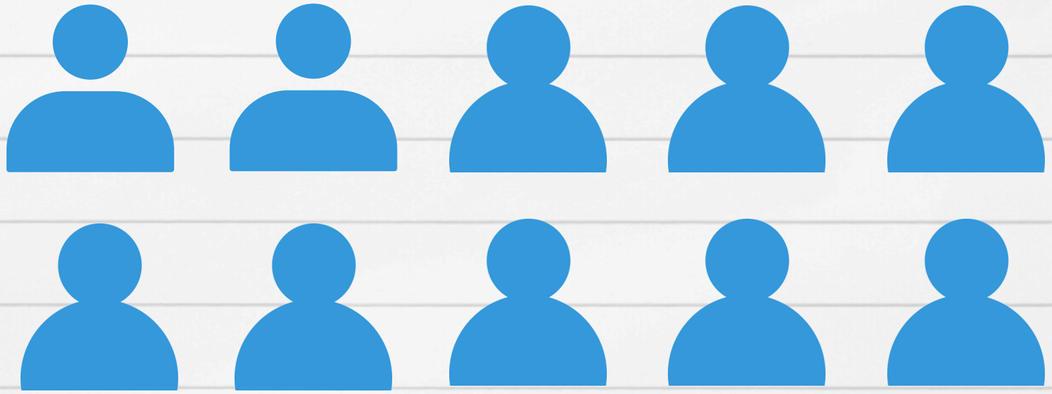
2) Randomly arrange the top scorers into teams



RANDOM CATEGORICALTEAMGENERATOR



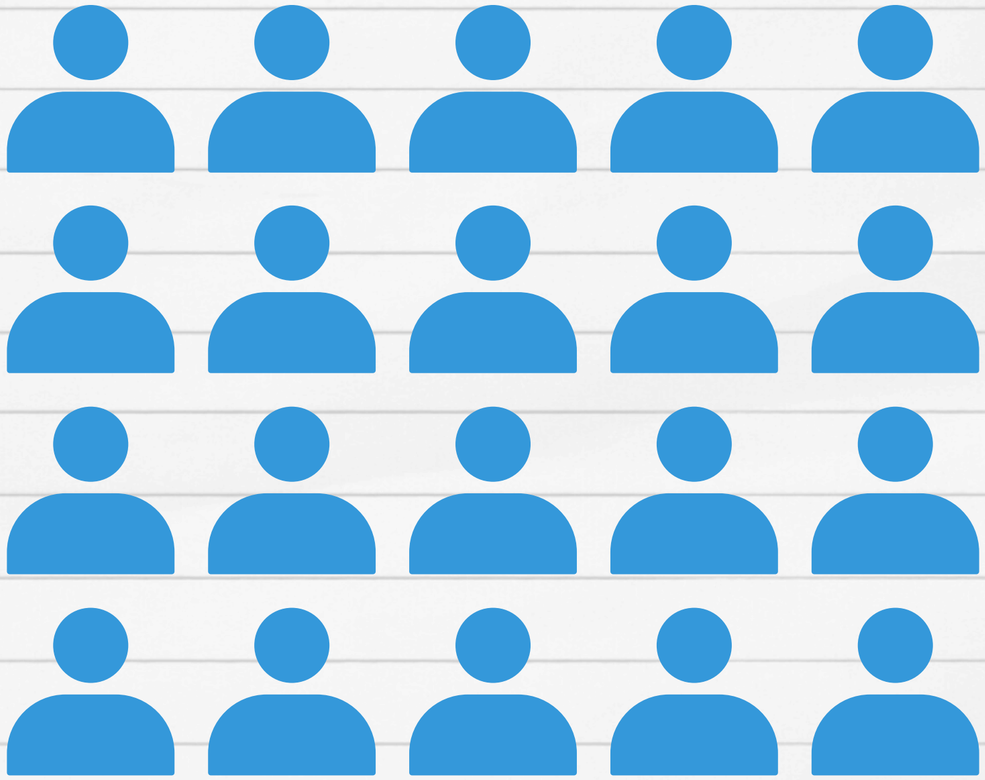
RANDOM CATEGORICALTEAMMAKER



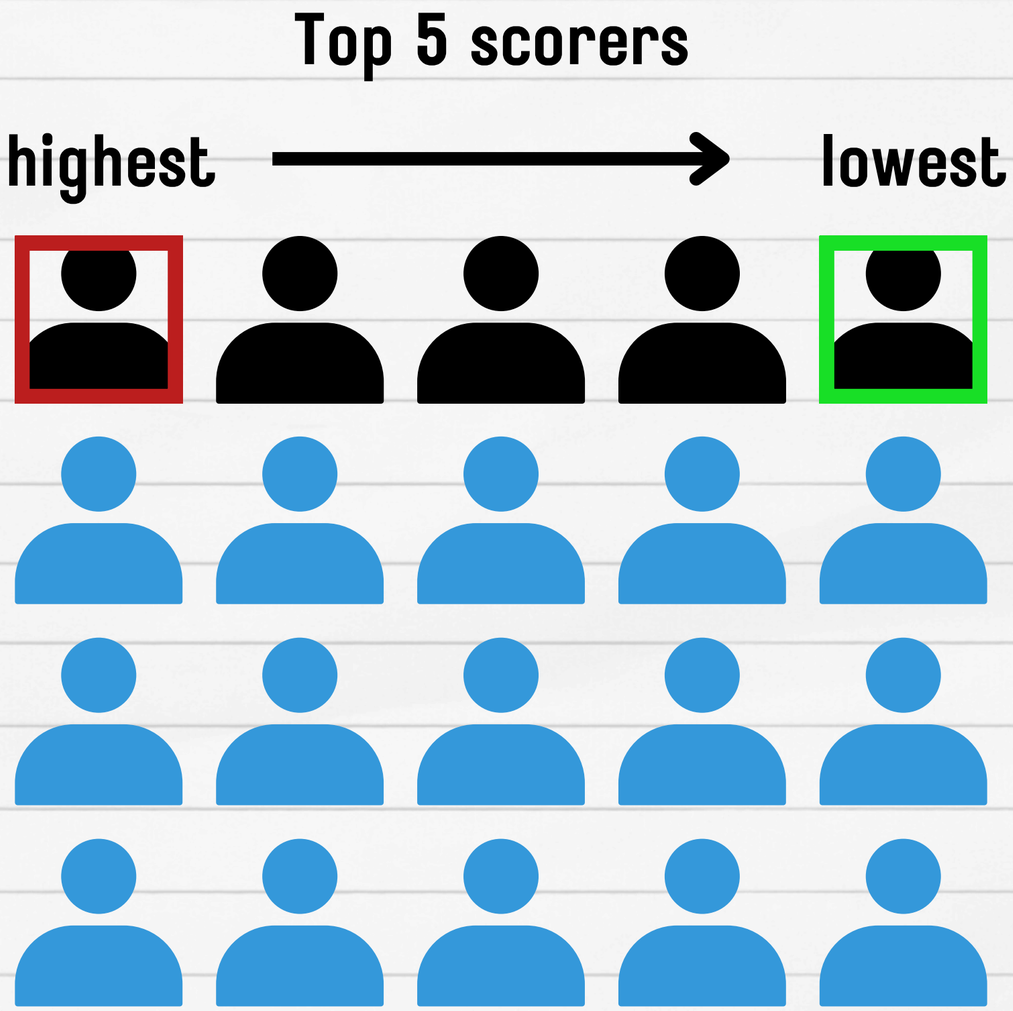
```cpp
// Class to create random teams with categories
class RandomCategoricalTeamGenerator : public TeamGenerator {
public:
    vector<Team> createTeams(int numTeams) {
        // Sort Persons based on their scores in descending order
        sort(Persons.begin(), Persons.end(), [](const Person& s1, const Person& s2) {
            return s1.getScore() > s2.getScore();
        });

        teams.clear();
        teams.resize(numTeams);

        // Assign each Person to a random team in the order of their scores
        random_device rd;
        mt19937 gen(rd());
        for (const auto& Person : Persons) {
            uniform_int_distribution<> dis(0, numTeams - 1);
            int teamIndex = dis(gen);
            teams[teamIndex].addPerson(Person);
        }

        return teams;
    }
};
```

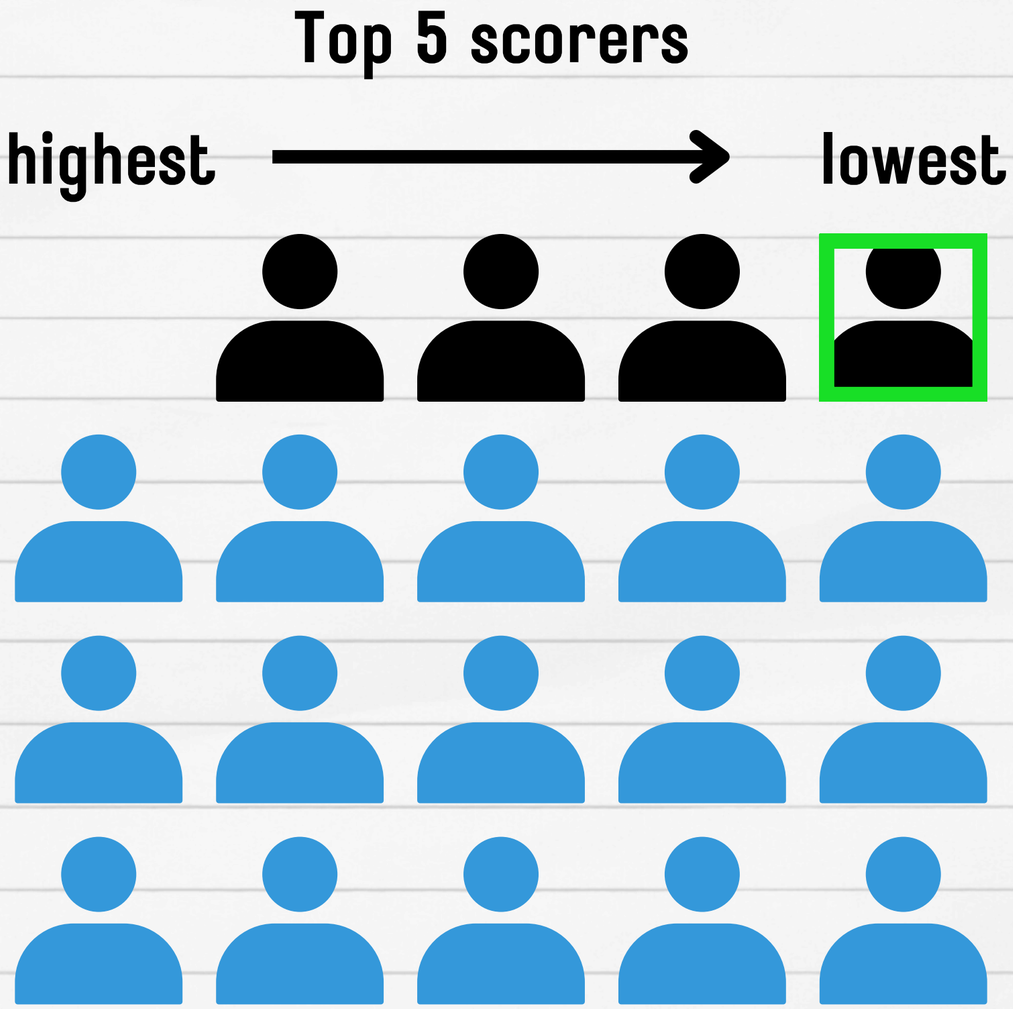
-TEST CASE-

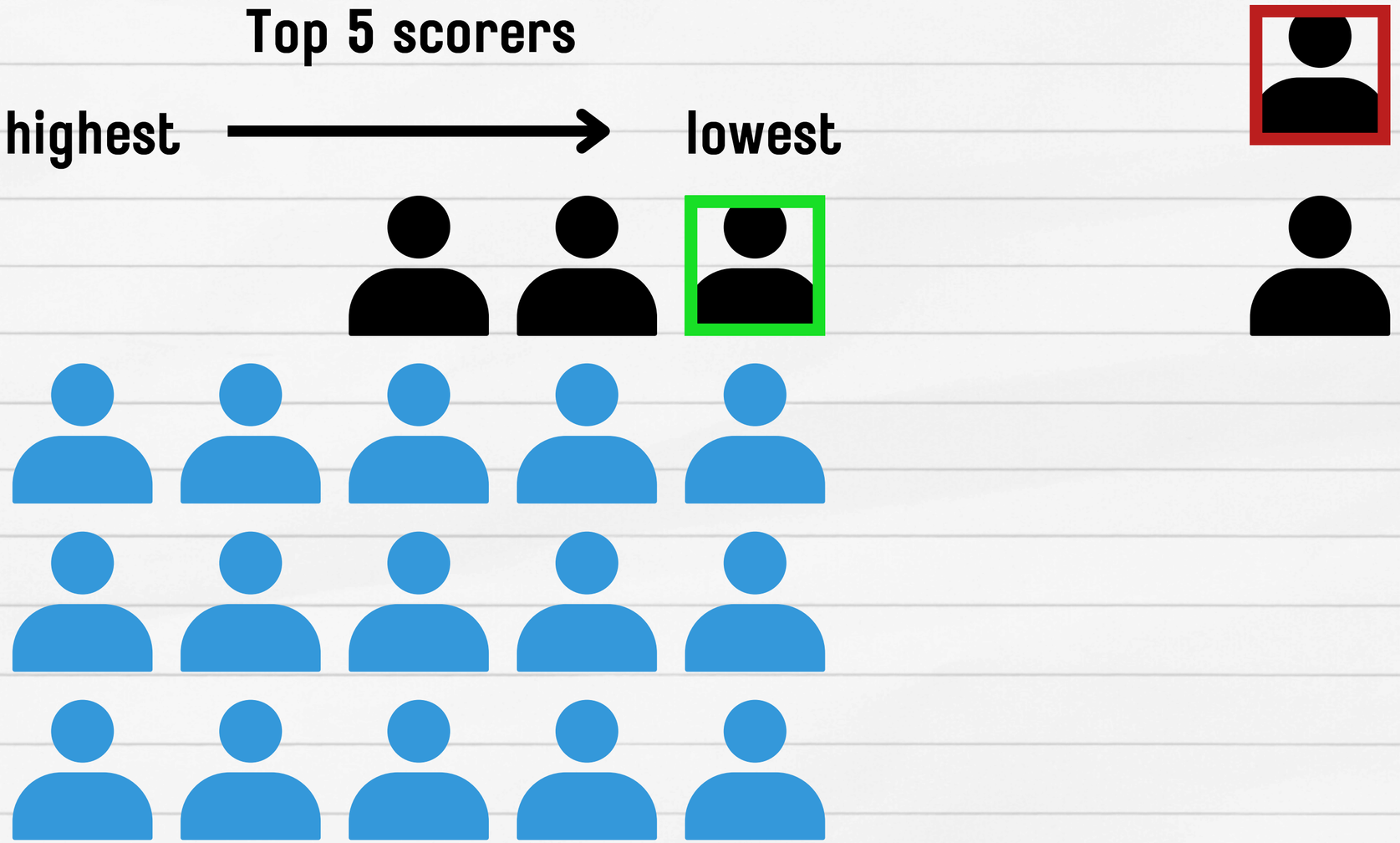
TEAMGENERATOR



Let’s say we have the following people to be sorted into groups:

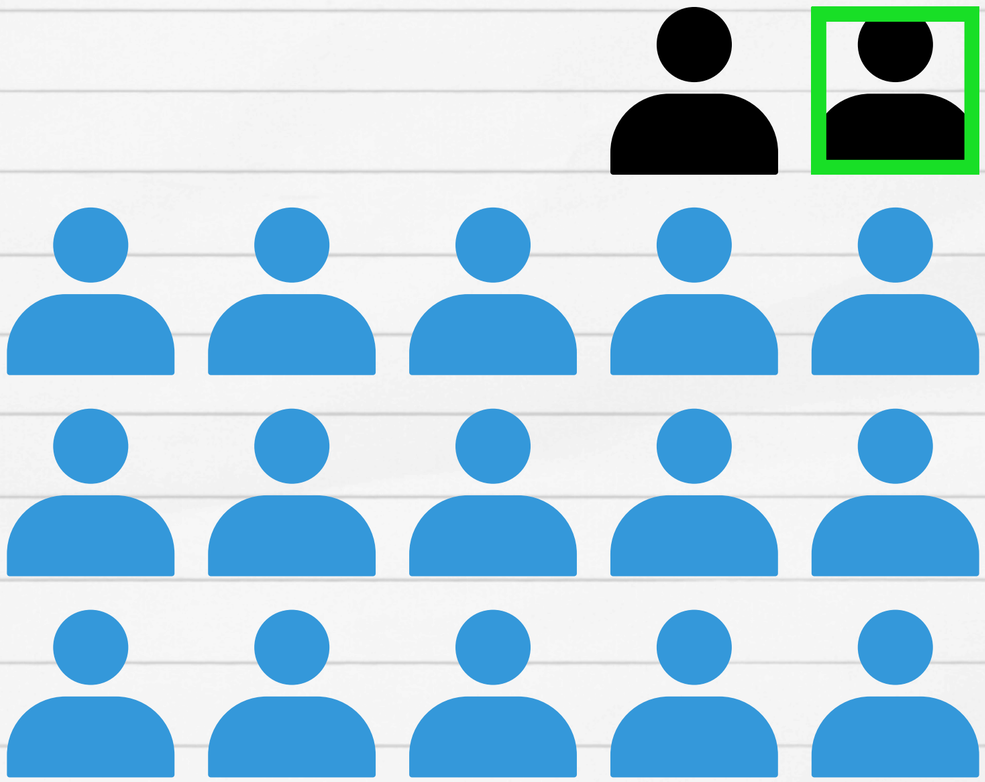

Top 5 scorers
highest
lowest

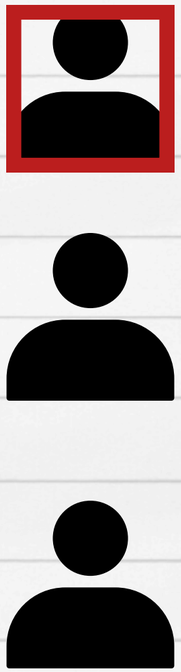

Top 5 scorers
highest
lowest


Top 5 scorers
highest
lowest

Top 5 scorers

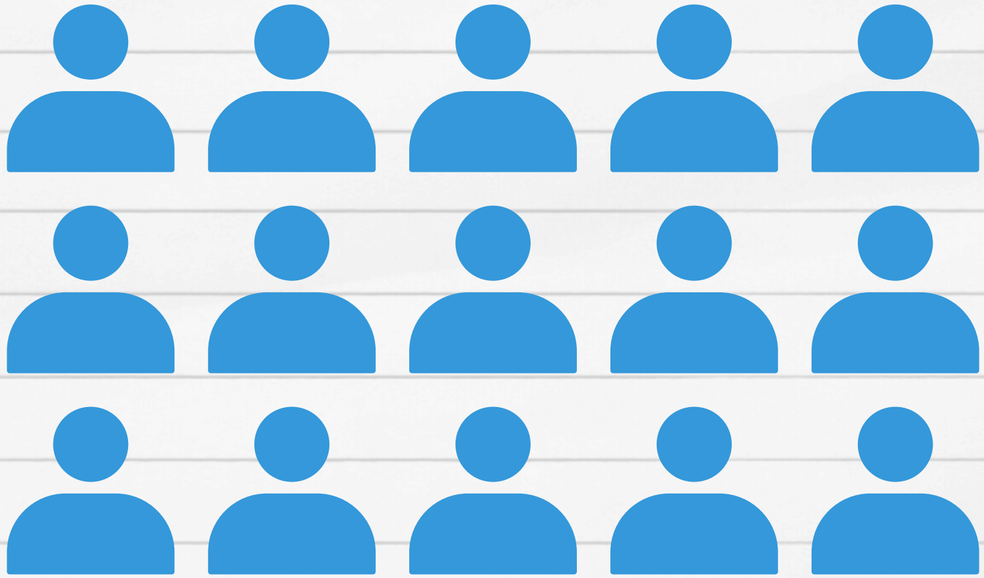
highest → lowest

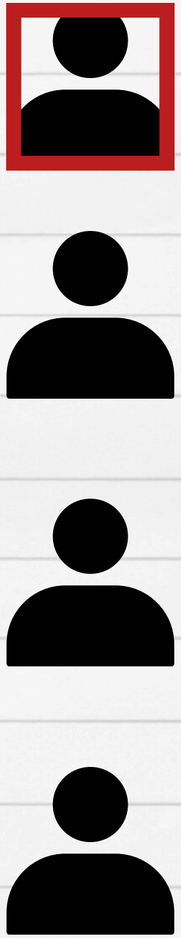




Top 5 scorers

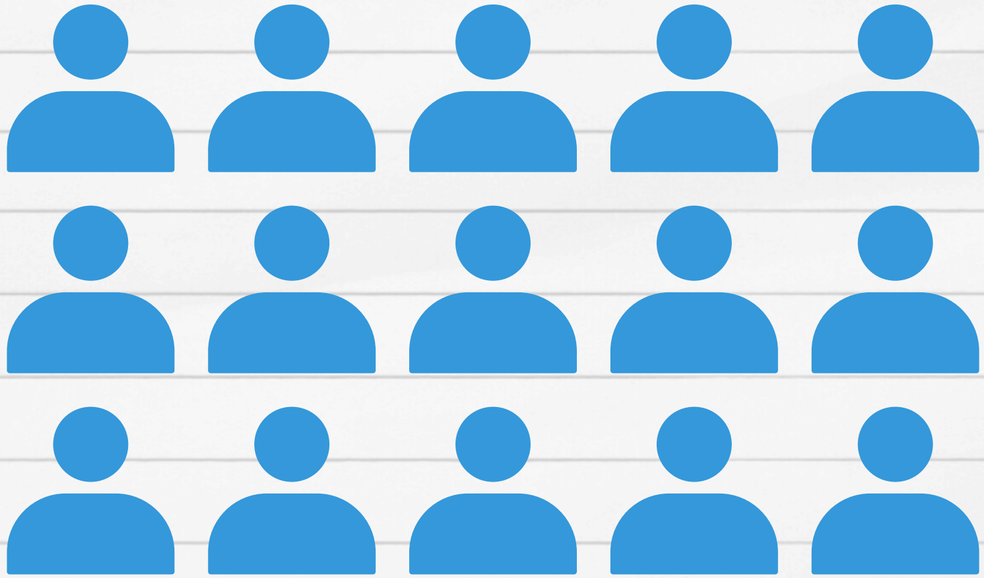
highest → lowest

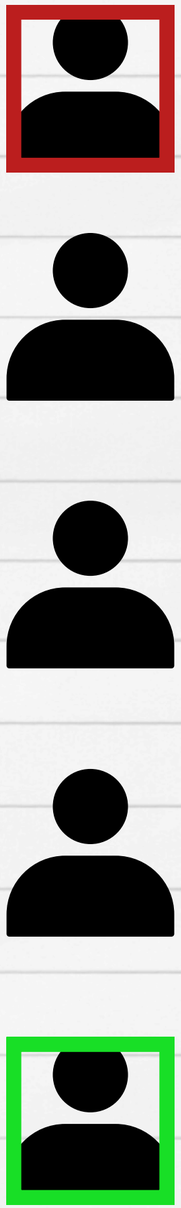




Top 5 scorers

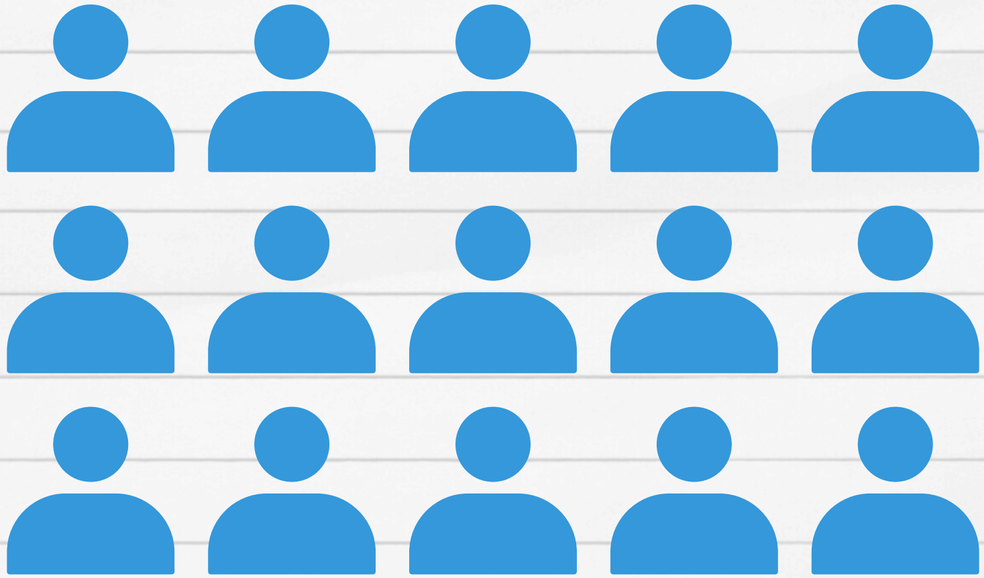
highest → lowest

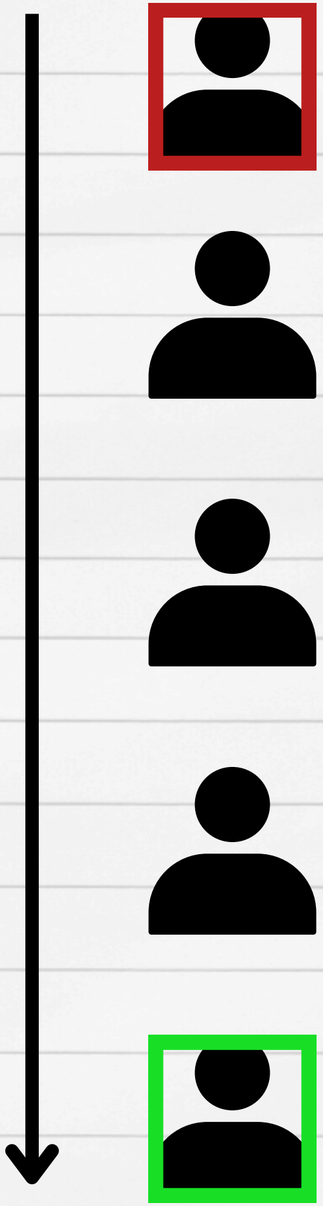




Top 5 scorers

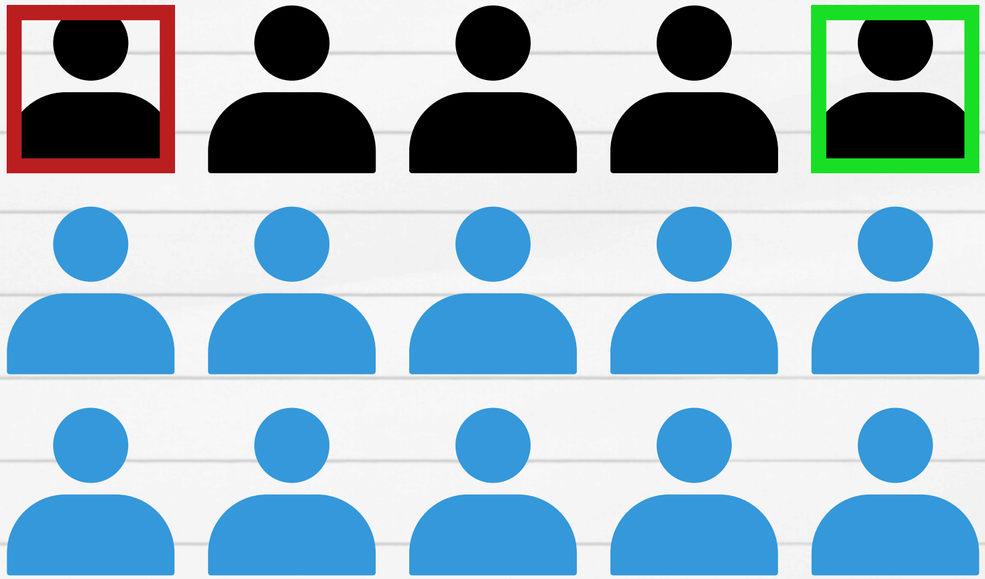
highest → lowest

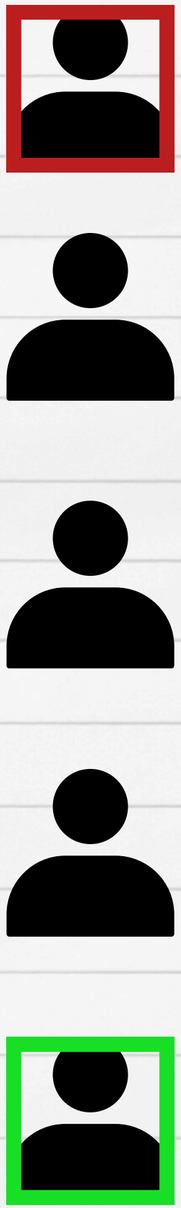




highest → lowest

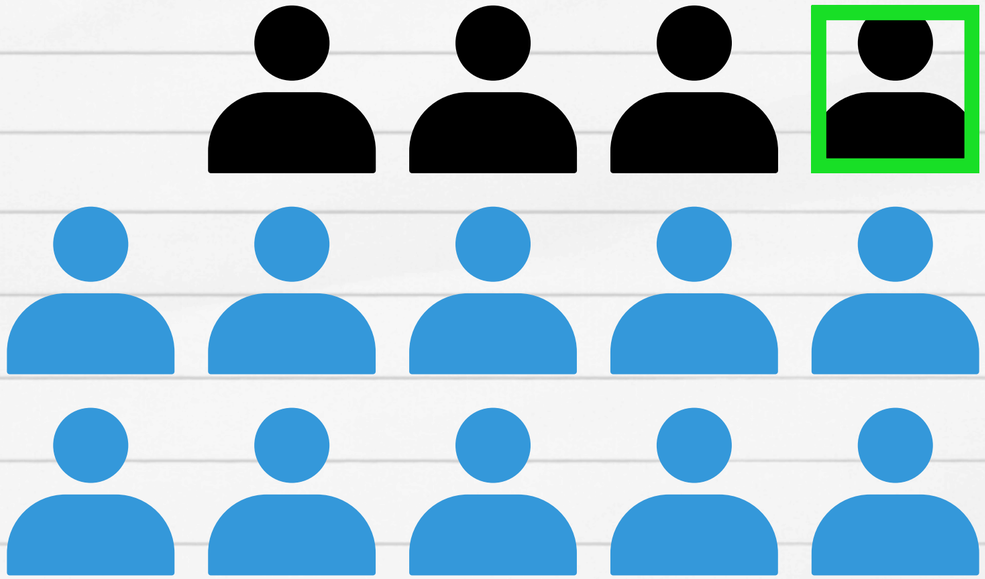
Next top 5 scorers

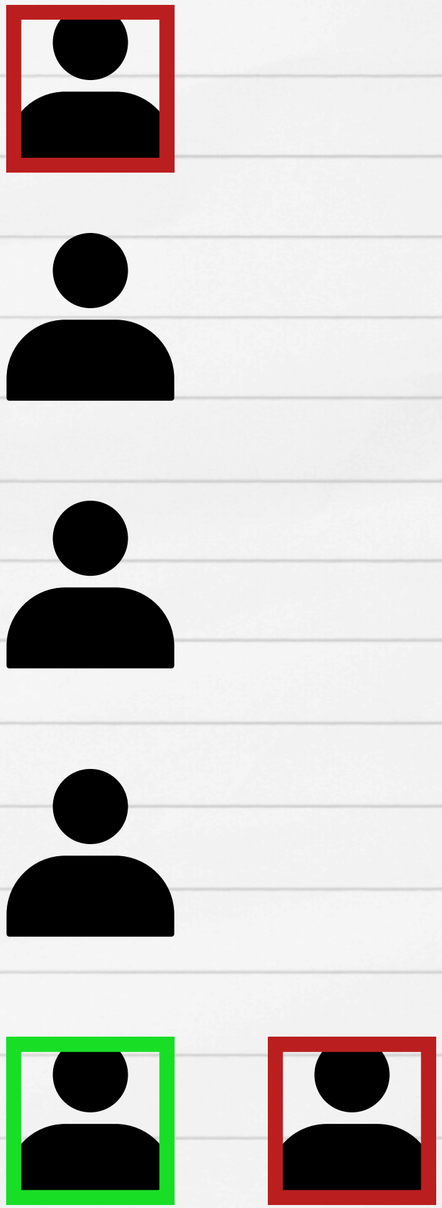




highest → lowest

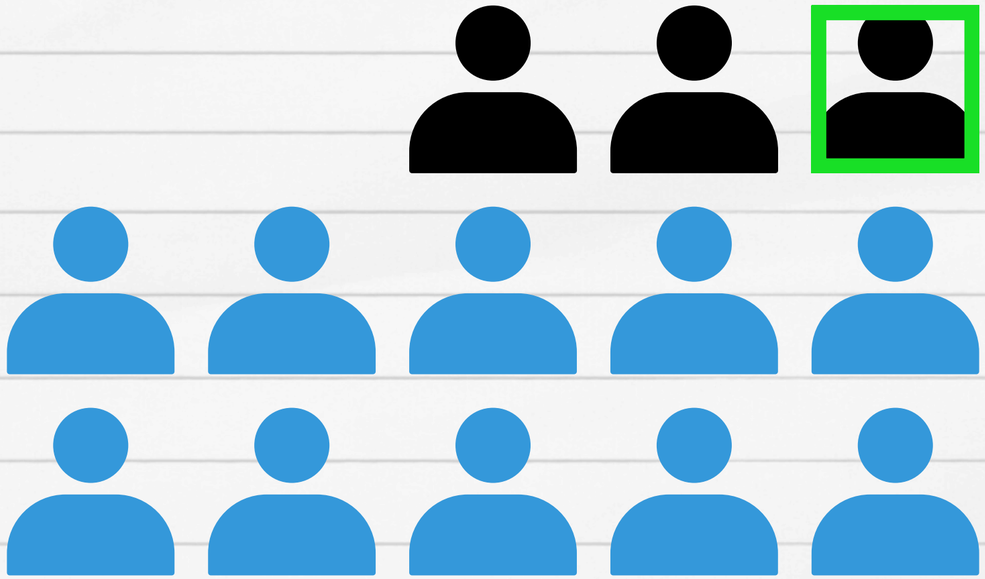
Next top 5 scorers

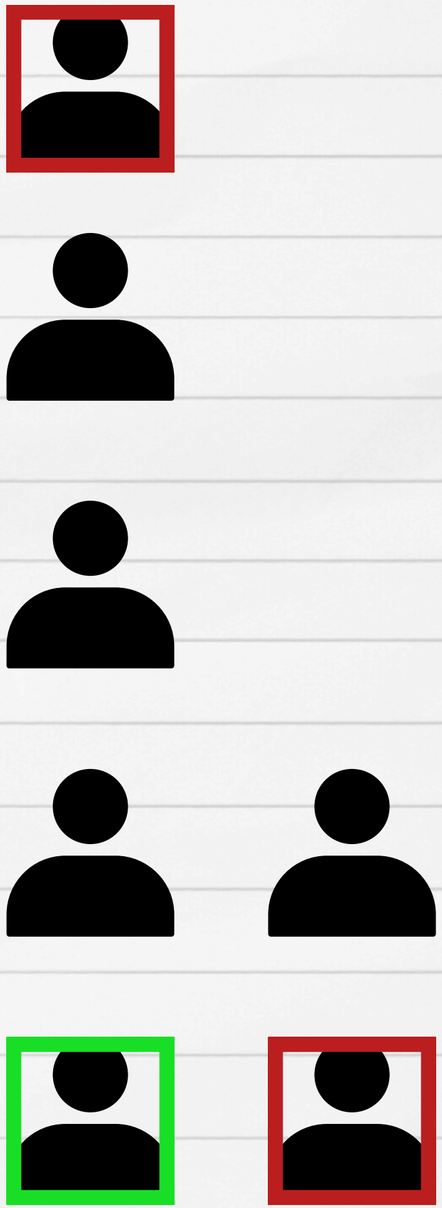




highest → lowest

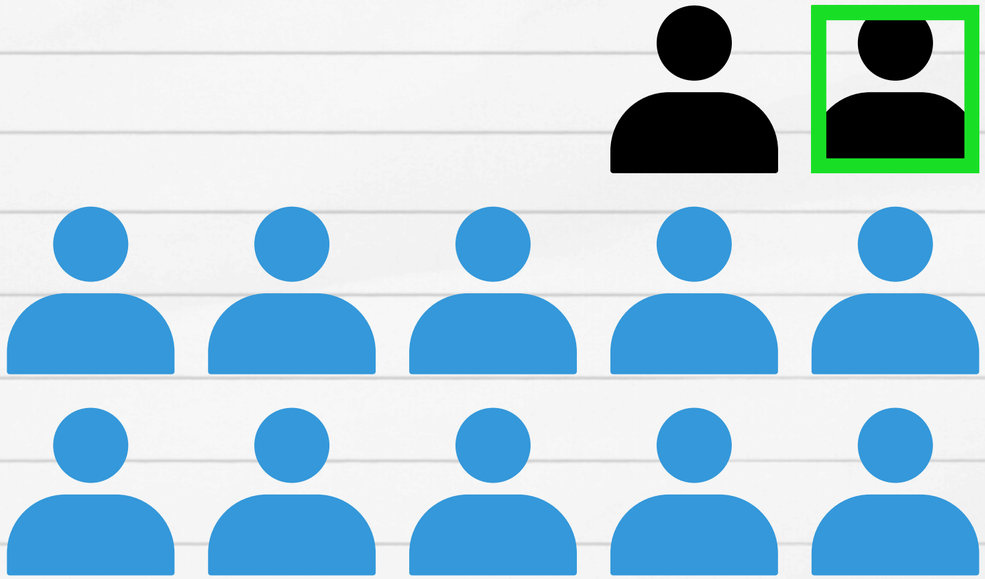
Next top 5 scorers

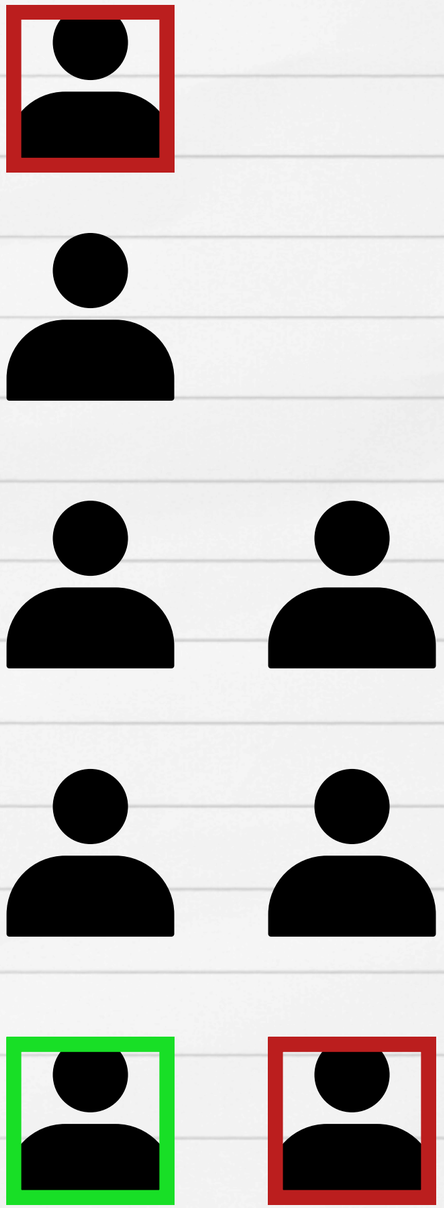




highest → lowest

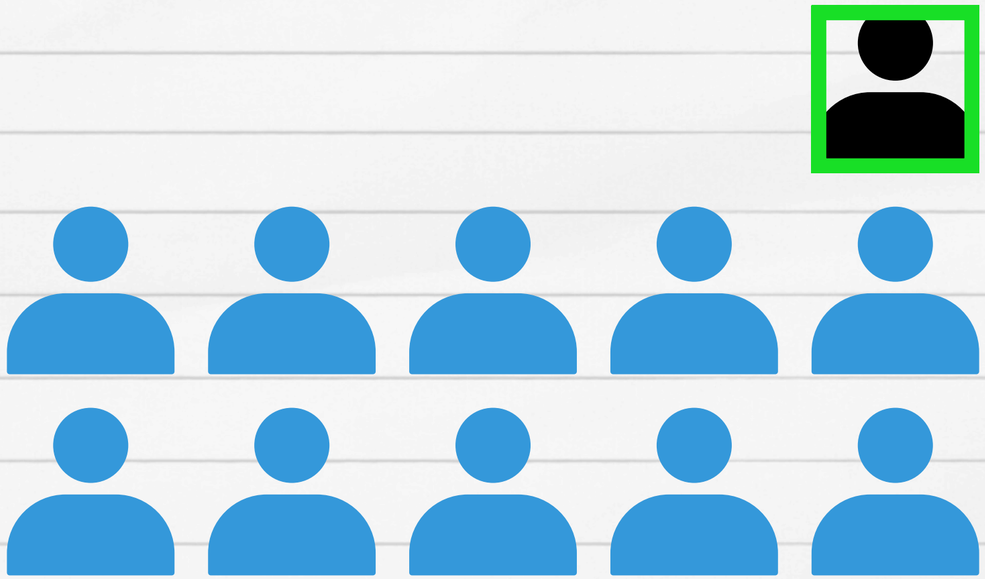
Next top 5 scorers

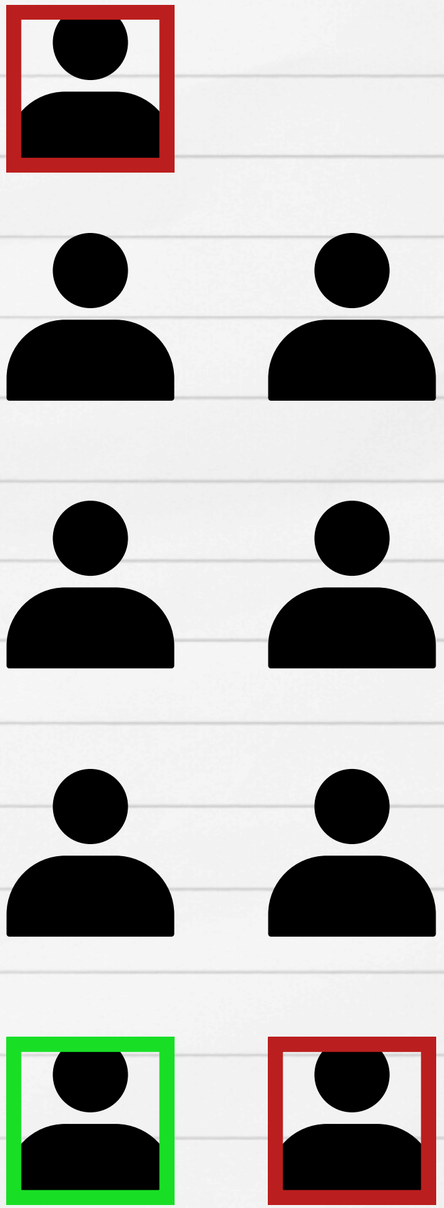




highest → lowest

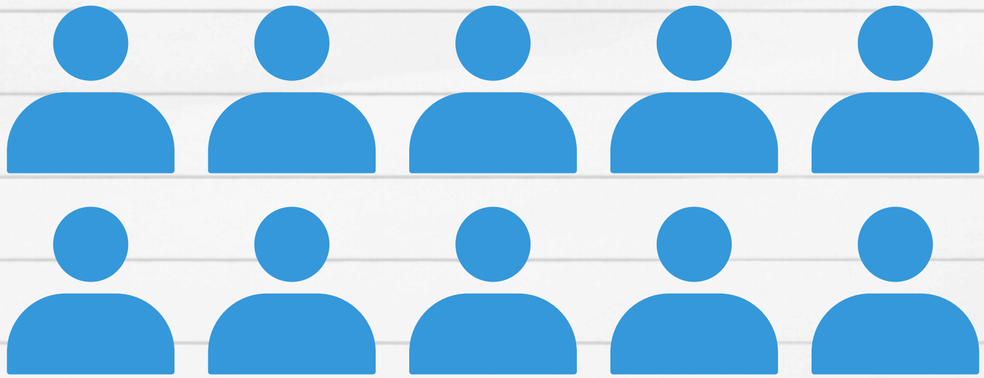
Next top 5 scorers

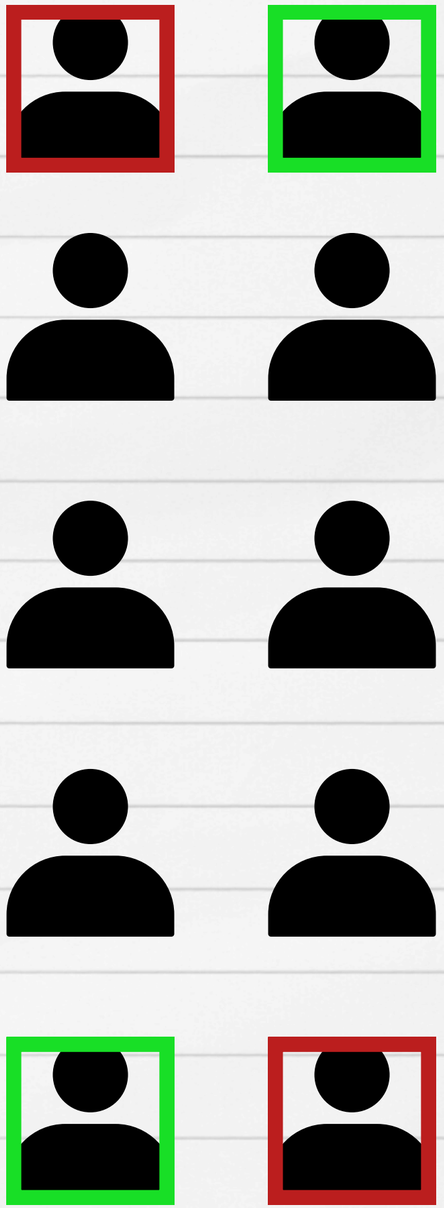




highest → lowest

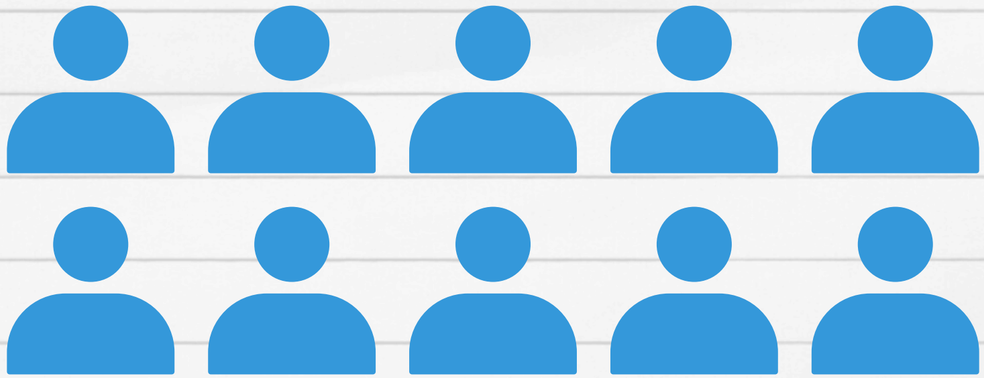
Next top 5 scorers

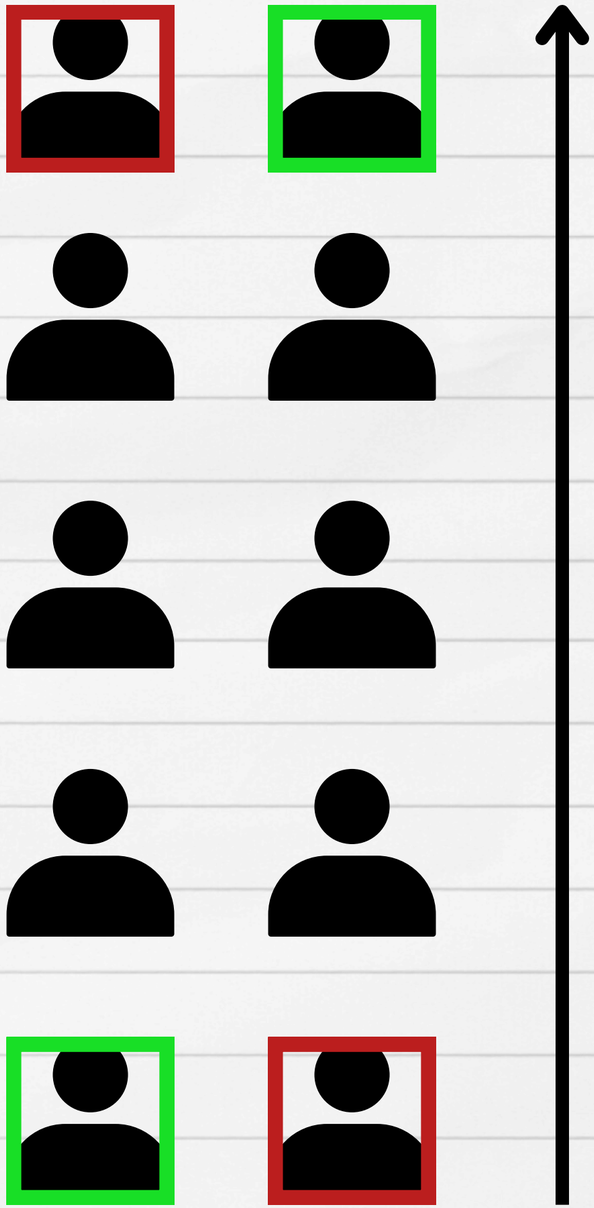




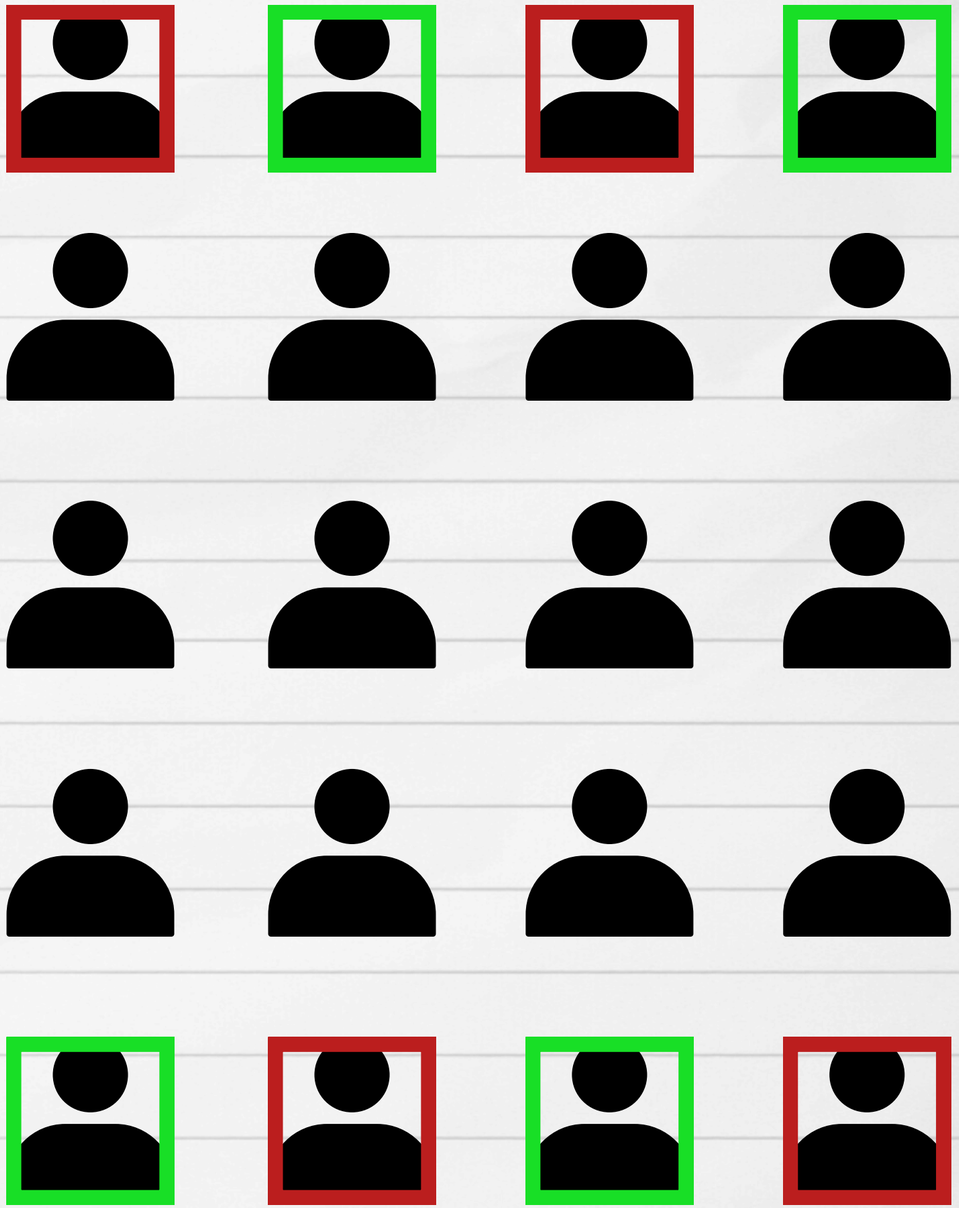
highest → lowest

Next top 5 scorers





AND SO WE GET SORTED TEAM:



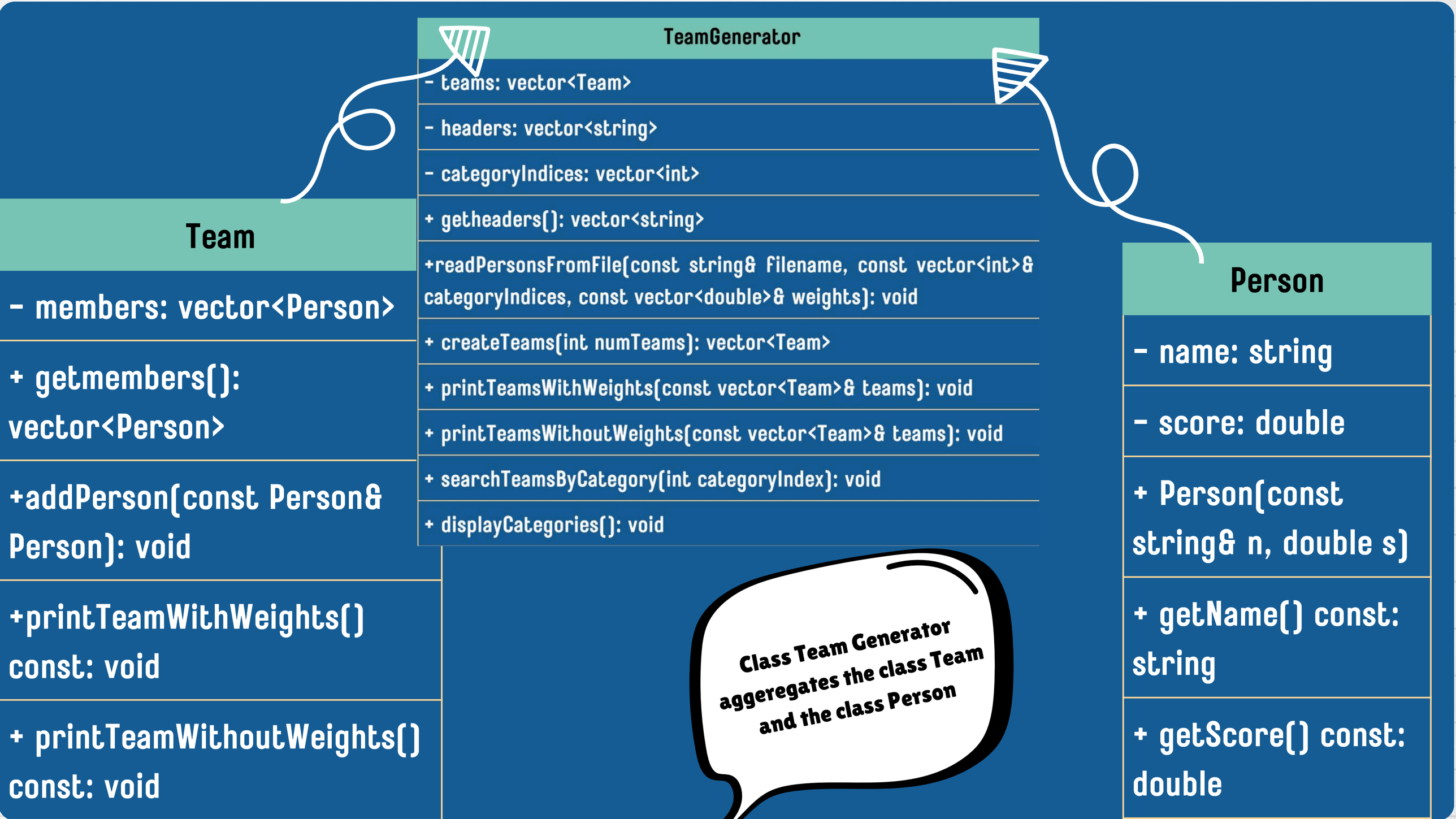
```cpp
void TeamGenerator::readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights) {
    string line;
    ifstream file(filename);

    //if file doesn't exist
     if (!file.is_open()) {
        cout << "Error: File \"" << filename << "\" doesn't exist." << endl;
        exit(0);
        return;
    }
    //if file is empty
    if (file.peek() == ifstream::traits_type::eof()) {
        cout << "Error: File \"" << filename << "\" is empty." << endl;
        exit(0);
        return;
    }

    // Get the number of categories from the header line
    getline(file, line);
    istringstream headerSS(line);
    string header;
    headers.clear(); // Clear the existing headers
    while (getline(headerSS, header, ',')) {
        headers.push_back(header);
    }
    int numCategories = headers.size() - 1; // -1 to exclude the "Name" header

    // Process the data lines
    Persons.clear(); // Clear the existing Persons
    while (getline(file, line)) {
        istringstream ss(line);
        string name;
        getline(ss, name, ',');

        vector<double> scores(numCategories, 0.0); // Initialize all scores to 0.0
        double weightedScore = 0.0;
        for (int i = 0; i < numCategories; ++i) {
            string scoreStr;
            getline(ss, scoreStr, ',');
            double score = stod(scoreStr);
            scores[i] = score;

            // Check if the current category is selected
            auto categoryIndex = find(categoryIndices.begin(), categoryIndices.end(), i);
```

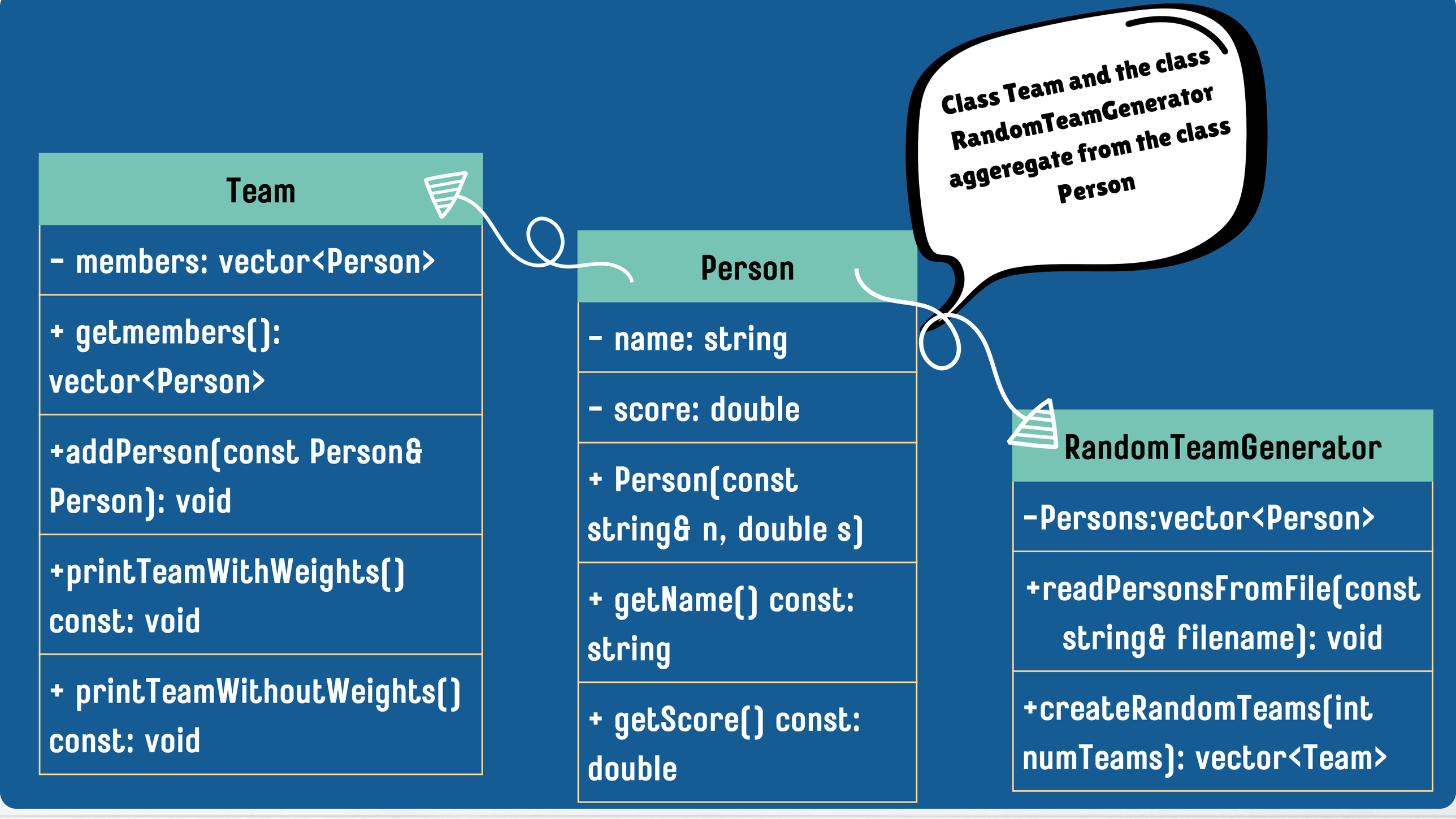
```
            // Check if the current category is selected
            auto categoryIndex = find(categoryIndices.begin(), categoryIndices.end(), i);
            if (categoryIndex != categoryIndices.end()) {
                int weightIndex = categoryIndex - categoryIndices.begin();
                weightedScore += score * weights[weightIndex];
            }
        }

        Persons.push_back(Person(name, weightedScore));
    }

    this->categoryIndices = categoryIndices; // Store the category indices
}
```

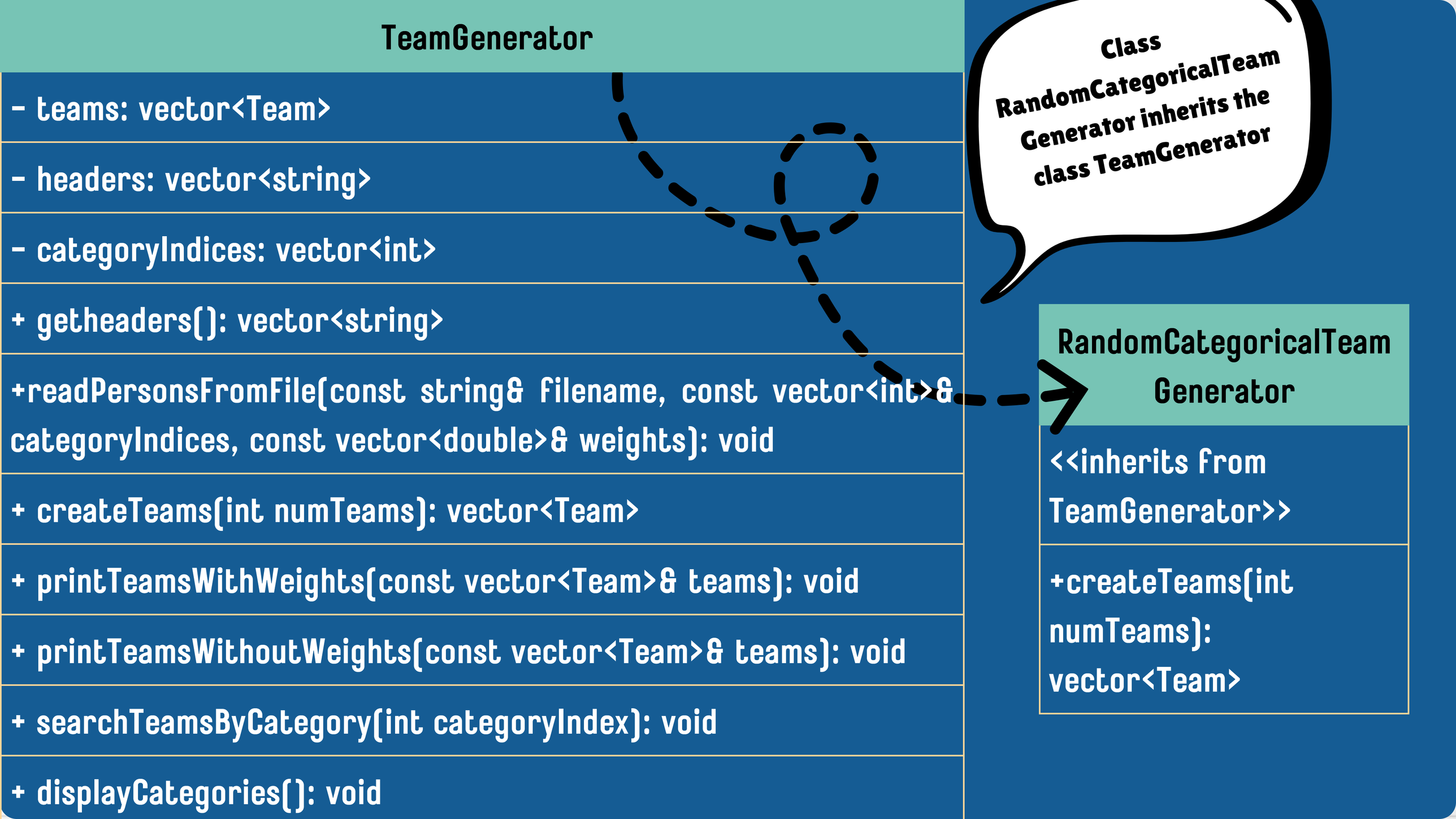
```cpp
vector<Team> TeamGenerator::createTeams(int numTeams) {
    // A priority queue to store Persons sorted by their scores in descending order
    priority_queue<Person, vector<Person>, PersonComparator> PersonPriorityQueue(Persons.begin(), Persons.end());

    teams.clear();
    teams.resize(numTeams);

    int currentTeam = 0;
    bool forward = true;

    while (!PersonPriorityQueue.empty()) {
        teams[currentTeam].addPerson(PersonPriorityQueue.top());
        PersonPriorityQueue.pop();

        // Update the currentTeam in a zig-zag pattern
        if (forward) {
            currentTeam++;
            if (currentTeam == numTeams) {
                forward = false;
                currentTeam = numTeams - 1;
            }
        } else {
            currentTeam--;
            if (currentTeam == -1) {
                forward = true;
                currentTeam = 0;
            }
        }
    }

    return teams;
}
```

-TEST CASE-

-UML-


TeamGenerator
- teams: vector<Team>
- headers: vector<string>
- categoryIndices: vector<int>
+ getheaders(): vector<string>
+readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights): void
+ createTeams(int numTeams): vector<Team>
+ printTeamsWithWeights(const vector<Team>& teams): void
+ printTeamsWithoutWeights(const vector<Team>& teams): void
+ searchTeamsByCategory(int categoryIndex): void
+ displayCategories(): void
Team
- members: vector<Person>
+ getmembers(): vector<Person>
+addPerson(const Person& Person): void
+printTeamWithWeights() const: void
+ printTeamWithoutWeights() const: void
Person
- name: string
- score: double
+ Person(const string& n, double s)
+ getName() const: string
+ getScore() const: double
Class Team Generator aggeregates the class Team and the class Person


Team
- members: vector<Person>
+ getmembers(): vector<Person>
+addPerson(const Person& Person): void
+printTeamWithWeights() const: void
+ printTeamWithoutWeights() const: void
Person
- name: string
- score: double
+ Person(const string& n, double s)
+ getName() const: string
+ getScore() const: double
Class Team and the class RandomTeamGenerator aggeregate from the class Person
RandomTeamGenerator
-Persons:vector<Person>
+readPersonsFromFile(const string& filename): void
+createRandomTeams(int numTeams): vector<Team>


TeamGenerator
- teams: vector<Team>
- headers: vector<string>
- categoryIndices: vector<int>
+ getheaders(): vector<string>
+readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights): void
+ createTeams(int numTeams): vector<Team>
+ printTeamsWithWeights(const vector<Team>& teams): void
+ printTeamsWithoutWeights(const vector<Team>& teams): void
+ searchTeamsByCategory(int categoryIndex): void
+ displayCategories(): void
Class RandomCategoricalTeam Generator inherits the class TeamGenerator
RandomCategoricalTeam Generator
<<inherits from TeamGenerator>>
+createTeams(int numTeams): vector<Team>

-THE OOPS-

Concepts we have Covered:

1) Inheritance
2) Encapsulation
4) Aggregation
5) Polymorphism

1) RUNTIME POLYMORPHISM AND INHERITANCE

Base class and Abstract Class:

```
class Generator {
protected:
    vector<Person> Persons;
    void virtual readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights) = 0;
};
```

Derived Classes:

```
class TeamGenerator : public Generator {
protected:
    vector<Team> teams;
    vector<string> headers;
    vector<int> categoryIndices;
public:
    vector<string> getheaders() { return headers; }
    void readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights);
    vector<Team> createTeams(int numTeams);
    void printTeamsWithWeights(const vector<Team>& teams);
    void printTeamsWithoutWeights(const vector<Team>& teams);
    void searchTeamsByCategory(int categoryIndex, const string& filename);
    void displayCategories();
};
```

```
class RandomTeamGenerator : public Generator {
public:
   void readPersonsFromFile(const string& filename, const vector<int>& categoryIndices = {}, const vector<double>& weights = {});
   vector<Team> createRandomTeams(int numTeams);
 };
```

Both the derived classes inherit vector Person <person>

from the base class(Class Generator) and override the function readPersonsFromFile:

Class RandomTeamGenerator :

```cpp
void RandomTeamGenerator::readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weight){
    string line;
    ifstream file(filename);

    // Skip the header line
    getline(file, line);

    // Process the data lines
    Persons.clear(); // Clear the existing Persons
    while (getline(file, line)) {
        istringstream ss(line);
        string name;
        getline(ss, name, ',');
        Persons.push_back(Person(name, 0.0)); // Scores are not used for random teams
    }
}
```

Class TeamGenerator:

```cpp
void TeamGenerator::readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights) {
    string line;
    ifstream file(filename);

    //if file doesn't exist
     if (!file.is_open()) {
        cout << "Error: File \"" << filename << "\" doesn't exist." << endl;
        exit(0);
        return;
    }
    //if file is empty
    if (file.peek() == ifstream::traits_type::eof()) {
        cout << "Error: File \"" << filename << "\" is empty." << endl;
        exit(0);
        return;
    }

    // Get the number of categories from the header line
    getline(file, line);
    istringstream headerSS(line);
    string header;
    headers.clear(); // Clear the existing headers
    while (getline(headerSS, header, ',')) {
        headers.push_back(header);
    }
    int numCategories = headers.size() - 1; // -1 to exclude the "Name" header

    // Process the data lines
    Persons.clear(); // Clear the existing Persons
    while (getline(file, line)) {
        istringstream ss(line);
        string name;
        getline(ss, name, ',');

        vector<double> scores(numCategories, 0.0); // Initialize all scores to 0.0
        double weightedScore = 0.0;
        for (int i = 0; i < numCategories; ++i) {
            string scoreStr;
            getline(ss, scoreStr, ',');
            double score = stod(scoreStr);
            scores[i] = score;

            // Check if the current category is selected
            auto categoryIndex = find(categoryIndices.begin(), categoryIndices.end(), i);
            if (categoryIndex != categoryIndices.end()) {
                int weightIndex = categoryIndex - categoryIndices.begin();
                weightedScore += score * weights[weightIndex];
            }
        }

        Persons.push_back(Person(name, weightedScore));
    }

    this->categoryIndices = categoryIndices; // Store the category indices
}
```

2) ENCAPSULATION

```
class Student {
protected:
    string name;
    double score;
public:
```

```
class TeamGenerator : public Generator {
protected:
    vector<Team> teams;
    vector<string> headers;
    vector<int> categoryIndices;
```

3) AGGREGATION

```
class Team {
protected:
    vector<Person> members;
```

```
class TeamGenerator : public Generator {
protected:
    vector<Team> teams;
    vector<string> headers;
    vector<int> categoryIndices;
```

```
class RandomCategoricalTeamGenerator : public TeamGenerator {
public:
    vector<Team> createTeams(int numTeams) {
```

-DATA STRUCTURES-

1) Vector
2) Map
3) Priority Queue

1) VECTOR

```
vector<Student> members;
```

```
vector<Team> teams;
vector<string> headers;
vector<int> categoryIndices;
```

We used Vectors to be able to store information from the CSV file and also store information after the algorithm ran such as the teams themselves or the calculation of the final score

Vectors were used over arrays due to being dynamic in nature

Vectors were used over Linked List due to our code requiring both a sequential and random access iterative approach. For the random access iterative approach, the complexity in a vector is O(1) while it is O(N) in a list

2) MAP

```cpp
// Read the scores from the file
map<string, double> PersonScores;
ifstream file(filename);
if (file.is_open()) {
    string line;
    getline(file, line); // Skip the header row

    while (getline(file, line)) {
        istringstream ss(line);
        string name;
        vector<double> scores;
        getline(ss, name, ',');
        string scoreStr;
        while (getline(ss, scoreStr, ',')) {
            scores.push_back(stod(scoreStr));
        }
        PersonScores[name] = scores[categoryIndex + 1]; // Assuming 0-based category index
    }
    file.close();
} else {
    cout << "Unable to open the file!" << endl;
    return;
}

// Create a map to store teams and their total scores in the selected category
map<double, vector<int>, greater<double>> teamScores;
for (int team = 0; team < teams.size(); ++team) {
    double totalScore = 0.0;
    for (const auto& Person : teams[team].getmembers()) {
        totalScore += PersonScores[Person.getName()];
    }
    teamScores[totalScore].push_back(team);
}

cout << "Teams sorted by total score in " << headers[categoryIndex + 1] << " category (highest to lowest):" << endl;
int rank = 1;
for (const auto& teamScore : teamScores) {
    // Sort the teams with the same score using bubble sort
    vector<int> sortedTeams = teamScore.second;
    for (int i = 0; i < sortedTeams.size() - 1; ++i) {
        for (int j = 0; j < sortedTeams.size() - i - 1; ++j) {
            if (teams[sortedTeams[j]].getmembers().size() < teams[sortedTeams[j + 1]].getmembers().size()) {
                swap(sortedTeams[j], sortedTeams[j + 1]);
            }
        }
    }
```

This map has:

- Keys: double (representing total scores)
- Values: vector<int> (storing indices of teams with the same total score)
- Ordering: Keys are ordered by greater<double>, meaning highest scores come first (descending order).
- Purpose of the map: This teamScores map helps store information about teams in a specific category. It associates a team‘s total score (key) with a vector of indices (value) for all teams that share that same score.

2) MAP

```
// Read the scores from the file
map<string, double> PersonScores;
ifstream file(filename);
if (file.is_open()) {
    string line;
    getline(file, line); // Skip the header row

    while (getline(file, line)) {
        istringstream ss(line);
        string name;
        vector<double> scores;
        getline(ss, name, ',');
        string scoreStr;
        while (getline(ss, scoreStr, ',')) {
            scores.push_back(stod(scoreStr));
        }
        PersonScores[name] = scores[categoryIndex + 1]; // Assuming 0-based category index
    }
    file.close();
} else {
    cout << "Unable to open the file!" << endl;
    return;
}

// Create a map to store teams and their total scores in the selected category
map<double, vector<int>, greater<double>> teamScores;
for (int team = 0; team < teams.size(); ++team) {
    double totalScore = 0.0;
    for (const auto& Person : teams[team].getmembers()) {
        totalScore += PersonScores[Person.getName()];
    }
    teamScores[totalScore].push_back(team);
}

cout << "Teams sorted by total score in " << headers[categoryIndex + 1] << " category (highest to lowest):" << endl;
int rank = 1;
for (const auto& teamScore : teamScores) {
    // Sort the teams with the same score using bubble sort
    vector<int> sortedTeams = teamScore.second;
    for (int i = 0; i < sortedTeams.size() - 1; ++i) {
        for (int j = 0; j < sortedTeams.size() - i - 1; ++j) {
            if (teams[sortedTeams[j]].getmembers().size() < teams[sortedTeams[j + 1]].getmembers().size()) {
                swap(sortedTeams[j], sortedTeams[j + 1]);
            }
        }
    }
```

- Efficient Sorting & Access: std::map stores teams with same score together (vector of indices). This allows easy sorting by team size and faster access by total score (key) compared to using separate data structures with std::pair.
- Faster Iteration for Score Range: std::map keeps scores sorted. This enables efficient iteration over teams within a specific score range by leveraging the map‘s structure, compared to additional filtering needed with std::pair.

3) PRIORITY QUEUE

```
vector<Team> TeamGenerator::createTeams(int numTeams) {
    // A priority queue to store Persons sorted by their scores in descending order
    priority_queue<Person, vector<Person>, PersonComparator> PersonPriorityQueue(Persons.begin(), Persons.end());

    teams.clear();
    teams.resize(numTeams);

    int currentTeam = 0;
    bool forward = true;

    while (!PersonPriorityQueue.empty()) {
        teams[currentTeam].addPerson(PersonPriorityQueue.top());
        PersonPriorityQueue.pop();

        // Update the currentTeam in a zig-zag pattern
        if (forward) {
            currentTeam++;
            if (currentTeam == numTeams) {
                forward = false;
                currentTeam = numTeams - 1;
            }
        } else {
            currentTeam--;
            if (currentTeam == -1) {
                forward = true;
                currentTeam = 0;
            }
        }
    }

    return teams;
}
```

By using a priority Queue, the students are automatically sorted into descending order which makes the code shorter and more efficient than storing it in a vector and then sorting it.

-CLASSES-

Student class

```
19      // Class to represent a student
20      class Student {
21      protected:
22          string name;
23          double score;
24      public:
25          Student(const string& n, double s);
26          string getName() const;
27          double getScore() const;
28      };
```

Method

- getName(): Returns the student‘s name.
- getScore(): Returns the student‘s score.

Team Class

```
// Class to represent a team
class Team {
protected:
    vector<Person> members;
public:
    vector<Person> getmembers() { return members; }
    void addPerson(const Person& Person);
    void printTeamWithWeights() const;
    void printTeamWithoutWeights() const;
    void printTeamWithoutWeights(ostream& os) const;
};
```

Method

- addStudent(const Student & student): Adds a student to the team.
- printTeamWithWeights(): Prints the team's members with their scores.
- printTeamWithoutWeights( ): Prints the team's members sorted by name.

Team-Generator class

```
// Class to create balanced teams
class TeamGenerator : public Generator {
protected:
    vector<Team> teams;
    vector<string> headers;
    vector<int> categoryIndices;
public:
    vector<string> getheaders() { return headers; }
    void readPersonsFromFile(const string& filename, const vector<int>& categoryIndices, const vector<double>& weights);
    vector<Team> createTeams(int numTeams);
    void printTeamsWithWeights(const vector<Team>& teams);
    void printTeamsWithoutWeights(const vector<Team>& teams);
    void searchTeamsByCategory(int categoryIndex, const string& filename);
    void displayCategories();
};
```

Method

- createTeams(int numTeams): Creates balanced teams.
- printTeamsWithWeights(): Prints teams with member scores.
- printTeamsWithoutWeights(): Prints teams without scores.
- searchTeamsByCategory(int categoryIndex): Searches and sorts teams by a specific category.
- displayCategories(): Displays available categories.

RandomCategoricalTeamGenerator class

```cpp
// Class to create random teams with categories
class RandomCategoricalTeamGenerator : public TeamGenerator {
public:
    vector<Team> createTeams(int numTeams) {
        // Sort Persons based on their scores in descending order
        sort(Persons.begin(), Persons.end(), [](const Person& s1, const Person& s2)
            return s1.getScore() > s2.getScore();
        });

        teams.clear();
        teams.resize(numTeams);

        // Assign each Person to a random team in the order of their scores
        random_device rd;
        mt19937 gen(rd());
        for (const auto& Person : Persons) {
            uniform_int_distribution<> dis(0, numTeams - 1);
            int teamIndex = dis(gen);
            teams[teamIndex].addPerson(Person);
        }

        return teams;
    }
};
```

Method

- Sorting by Scores: Ensures higher-scoring students are considered first, aiming for some balance in the teams.
- Random Assignment: Batches of students are assigned to a team randomly, ensuring unpredictability and fairness.
- Random Number Generation: Utilizes modern C++ random number generation techniques for true randomness.
- Scalability: The method can handle varying numbers of students and teams, making it versatile for different scenarios.

RandomTeamGenerator class

```cpp
// Class to create random teams
class RandomTeamGenerator : public Generator {
public:
    void readPersonsFromFile(const string& filename, const vector<int>& categoryIndices = {}, const vector<double>& weights = {});
    vector<Team> createRandomTeams(int numTeams);
};
```

Method

- Random Assignment: Each student is assigned to a team randomly.
- Random Number Generation: Utilizes modern C++ random number generation techniques for true randomness.

-SOME MAJOR FUNCTIONS-

```cpp
vector<Team> TeamGenerator::createTeams(int numTeams) {
    // A priority queue to store Persons sorted by their scores in descending order
    priority_queue<Person, vector<Person>, PersonComparator> PersonPriorityQueue(Persons.begin(), Persons.end());

    teams.clear();
    teams.resize(numTeams);

    int currentTeam = 0;
    bool forward = true;

    while (!PersonPriorityQueue.empty()) {
        teams[currentTeam].addPerson(PersonPriorityQueue.top());
        PersonPriorityQueue.pop();

        // Update the currentTeam in a zig-zag pattern
        if (forward) {
            currentTeam++;
            if (currentTeam == numTeams) {
                forward = false;
                currentTeam = numTeams - 1;
            }
        } else {
            currentTeam--;
            if (currentTeam == -1) {
                forward = true;
                currentTeam = 0;
            }
        }
    }

    return teams;
}
```

TeamGenerator: createTeams()

`TeamGenerator::createTeams()`

This function creates balanced teams from a list of `Person` objects using a priority queue initialized with the `Person` objects sorted by scores in descending order. The `teams` vector is cleared and resized to `numTeams`. The function uses a zig-zag assignment method where it iterates through the priority queue, adding the top `Person` to the current team, and then adjusts the current team index either forward or backward until all `Person` objects are assigned, resulting in balanced teams.

RandomTeamGenerator::createRandomTeams()

```
vector<Team> RandomTeamGenerator::createRandomTeams(int numTeams) {
    random_device rd;
    mt19937 gen(rd());

    vector<Team> teams(numTeams);
    int PersonIndex = 0;
    for (const auto& Person : Persons) {
        int teamIndex = PersonIndex % numTeams;
        teams[teamIndex].addPerson(Person);
        PersonIndex++;
    }

    return teams;
}
```

RandomCategoricalTeamGenerator::createTeams()

```cpp
// Class to create random teams with categories
class RandomCategoricalTeamGenerator : public TeamGenerator {
public:
    vector<Team> createTeams(int numTeams) {
        // Sort Persons based on their scores in descending order
        sort(Persons.begin(), Persons.end(), [](const Person& s1, const Person& s2) {
            return s1.getScore() > s2.getScore();
        });

        teams.clear();
        teams.resize(numTeams);

        // Assign each Person to a random team in the order of their scores
        random_device rd;
        mt19937 gen(rd());
        for (const auto& Person : Persons) {
            uniform_int_distribution<> dis(0, numTeams - 1);
            int teamIndex = dis(gen);
            teams[teamIndex].addPerson(Person);
        }

        return teams;
    }
};
```

`RandomCategoricalTeamGenerator::createTeams()`

This function creates random teams while considering categories and scores. It sorts the `Persons` list by scores in descending order, clears and resizes the `teams` vector to `numTeams`, and assigns each `Person` to a random team using a uniform distribution. The function returns the teams with the `Person` objects assigned in a manner that introduces randomness while maintaining consideration of scores.

`TeamGenerator::printTeamsWithoutWeights()`

```cpp
void TeamGenerator::printTeamsWithoutWeights(const vector<Team>& teams) {
    int subChoice;
    do {
    cout << "Sub-Menu:\n";
    cout << "1. Print teams to console\n";
    cout << "2. Write teams to a file\n";
    cout << "0. Return to previous menu\n";
    cout << "Enter your choice: ";
    cin >> subChoice;
    if (subChoice == 1) {
            for (int team = 0; team < teams.size(); ++team) {
                cout << "Team " << team + 1 << ": ";
                teams[team].printTeamWithoutWeights();
                cout << endl;
            }
        } else if (subChoice == 2) {
            string filename;
            cout << "Enter the filename: ";
            cin >> filename;

            ofstream file(filename);
            if (file.is_open()) {
                for (int team = 0; team < teams.size(); ++team) {
        if (file.is_open()) {
            for (int team = 0; team < teams.size(); ++team) {
                file << "Team " << team + 1 << ": ";
                teams[team].printTeamWithoutWeights(file);
                file << endl;
            }
            file.close();
            cout << "Teams written to " << filename << " successfully!" << endl;
        } else {
            cout << "Unable to open the file!" << endl;
        }
    }
} while (subChoice != 0);
```

`TeamGenerator::printTeamsWithoutWeights()`

This function provides a sub-menu for printing teams without weights. It offers options to print teams to the console or write them to a file. If the user chooses to print to the console, it iterates through the `teams` vector and prints each team. If the user chooses to write to a file, it prompts for a filename, opens the file, writes each team to the file, and closes the file. The sub-menu loops until the user chooses to return to the previous menu, allowing repeated operations.

```cpp
void TeamGenerator::searchTeamsByCategory(int categoryIndex, const string& filename) {
    // Check if the selected category was used to create the teams
    bool categoryUsed = false;
    for (int index : categoryIndices) {
        if (index == categoryIndex) {
            categoryUsed = true;
            break;
        }
    }

    if (!categoryUsed) {
        cout << "The selected category was not used to create the teams. Please choose another category." << endl;
        return;
    }

    // Read the scores from the file
    map<string, double> PersonScores;
    ifstream file(filename);
    if (file.is_open()) {
        string line;
        getline(file, line); // Skip the header row

        while (getline(file, line)) {
            istringstream ss(line);
      getline(file, line); // Skip the header row

      while (getline(file, line)) {
          istringstream ss(line);
          string name;
          vector<double> scores;
          getline(ss, name, ',');
          string scoreStr;
          while (getline(ss, scoreStr, ',')) {
              scores.push_back(stod(scoreStr));
          }
          PersonScores[name] = scores[categoryIndex + 1]; // Assuming 0-based category index
      }
      file.close();
  } else {
      cout << "Unable to open the file!" << endl;
      return;
```

TeamGenerator::searchTeamsByCategory()

`TeamGenerator::searchTeamsByCategory()`
This function searches teams based on a specific category and displays them sorted by their total scores in that category. It checks if the category is used by comparing the given `categoryIndex` with `categoryIndices`. Then, it reads `Person` scores from a file into a map. It calculates the total score for each team in the specified category and stores these scores in a map. Finally, it displays the teams sorted by their total scores in descending order, using a secondary sort by team size if scores are equal.

FRONT-END DEMONSTRATION