FEST MANAGEMENT SYSTEM

GROUP ASSIGNMENT
OOPS

STUDY GROUP - 7
(JuniHers)

Participant (Shambhavi)

```cpp
#include <iostream>
#include <vector>
using namespace std;

class Participant;
class Coordinator;
class Event {
private:
   string eventID;
   string name;
   string venue;
   Coordinator* eventCoordinator;
   vector<Participant*> participants;

public:

   Event(string eventID, string eventName) : eventID(eventID),
name(eventName) {}

   string getEventID() const {
       return eventID;
   }

   string getEventName() const {
       return name;
   }

   string getEventVenue() const {
       return venue;
   }

   void registerParticipant(Participant* participant) {
       participants.push_back(participant);
       // cout << "Participant " << participant->getName() << " registered
for event " << name << endl;
```

```
    }

    void displayParticipants() const {
        cout << "Participants for event " << name << ": ";
        for (const auto& participant : participants) {
            cout << participant->getName() << " ";
        }
        cout << endl;
    }
};

class Participant {
private:
    string participantID;
    string name;
    vector<Event*> eventsParticipated;

public:
    // Constructor
    Participant(string participantID, string name) :
participantID(participantID), name(name) {}

    void registerForEvent(Event* event) {
        // add participant's name in Event class list
        event->registerParticipant(this);
        eventsParticipated.push_back(event);
        cout << "Participant " << name << " registered for event " <<
event->getEventID() << endl;
    }

    string getParticipantID() const {
        return participantID;
    }

    string getName() const {
        return name;
    }

    void displayEventsParticipated() const {
        cout << "Events participated by " << name << ": ";
```

```
        for (const auto &event : eventsParticipated) {
            cout << event->getEventName() << " ";
        }
        cout << endl;
    }
};
```

Core Members (Mahua)

```
class CoreMember : public Fest {
private:
   vector<Coordinator> coordinators;
   string specialization;

public:
   CoreMember(const string &name, int id, const string &spec)
       : Fest(name, id), specialization(spec) {}

   // Method to check if a coordinator exists
   bool coordinatorExists(const Coordinator &coord) const {
       for (const auto &existingCoord : coordinators) {
           if (existingCoord.getID() == coord.getID()) {
               return true;
           }
       }
       return false;
   }

   // Method to add a coordinator to the list
   void addCoordinator(const Coordinator &coord) {
       if (!coordinatorExists(coord)) {
           coordinators.push_back(coord);
           cout << "Coordinator added successfully!" << endl;
       } else {
           cout << "Coordinator already exists!" << endl;
       }
   }

   // Method to display information
   void displayInfo() {
       cout << "Core Member Information:" << endl;
       Fest::displayInfo();
       cout << "Specialization: " << specialization << endl;
   }

   // Method to retrieve the list of coordinators
   const vector<Coordinator>& getCoordinators() const {
       return coordinators;
   }
};
```

Workforce (Tejal)

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Forward declaration
class Coordinator;

// Workforce class represents the students who assist in conducting events
during the fest.
class Workforce {
private:
    int id;                         // Unique identifier for the workforce
member
    string name;                // Name of the workforce member
    string eventID;             // Event assigned to the workforce member
    Coordinator* coordinator;       // Reference to the coordinator
supervising this workforce

public:
    // Constructor
    Workforce(int _id, const string& _name) : id(_id), name(_name),
coordinator(nullptr) {}

    // Method to assign a coordinator to the workforce member.
    void assignCoordinator(Coordinator* _coordinator) {
        coordinator = _coordinator;
    }
    // Method to complete a task for a specific event.
    void completeTask(const string& task) {
        cout << "Workforce member " << name << " completed task: " << task
<< " for event(s)." << endl;
    }
    // Getter method to retrieve the assigned coordinator.
    Coordinator* getCoordinator() const {
        return coordinator;
```

```
    }

    // Getter method to retrieve the name of the workforce member.
    string getName() const {
        return name;
    }
};

// Coordinator class represents the individuals responsible for managing
and coordinating the workforce.
class Coordinator {
private:
    string name;

public:
    // Constructor to initialize a coordinator with a name.
    Coordinator(const string& n) : name(n) {}

    // Add other methods as needed
};

int main() {
    Workforce workforce1(1, "John");
    Workforce workforce2(2, "Jane");

    Coordinator coordinator1("Coordinator 1");
    Coordinator coordinator2("Coordinator 2");

    // Assign coordinators to workforce members.
    workforce1.assignCoordinator(&coordinator1);
    workforce2.assignCoordinator(&coordinator2);

    // Get coordinator information for a workforce member.
    cout << workforce1.getName() << "'s coordinator: " <<
workforce1.getCoordinator() << endl;

    // Complete a task for a specific event.
    workforce1.completeTask("Setup booth");
    return 0;
}
```

(Department and integration) Disha

```
#include <bits/stdc++.h>
using namespace std;

class Participant;
class Coordinator;
class CoreMember;
class Event {
private:
  string eventID;
  string name;
  string venue;
  Coordinator* eventCoordinator;
  vector<Participant*> participants;
  vector<CoreMember*> corememberS;

public:


  Event(string eventID, string eventName) : eventID(eventID),
name(eventName) {}


  string getEventID() const {
      return eventID;
  }


  string getEventName() const {
      return name;
  }


  string getEventVenue() const {
      return venue;
```

```
    }


    void registerParticipant(Participant* participant) {
        participants.push_back(participant);
        // cout << "Participant " << participant->getName() << " registered
for event " << name << endl;
    }


    void displayParticipants() const {
        cout << "Participants for event " << name << ": ";
        for (const auto& participant : participants) {
            cout << participant->getName() << " ";
        }
        cout << endl;
    }

    void registerEventCoordinator(Coordinator* _coordinator) {
        if(eventCoordinator == NULL) eventCoordinator = _coordinator;
    }

     Coordinator* getCoordinator() {
         return eventCoordinator;
     }

    void registerEventCoremember(CoreMember* _coremember) {
       coremembers.push_back(_coremember);
    }

    void displayCoreMembers() const {
        cout << "Participants for event " << name << ": ";
        for (const auto& CoreMember : coremembers) {
            cout << CoreMember->getName() << " ";
        }
        cout << endl;
    }
};
```

```
class department {
   private:
     string name;
     vector<CoreMember*> coremembers;
   public:
     department(string _name) {
       name = _name;
     }

     void registerMember(CoreMember* _member) {
       coremembers.push_back(_member);
     }

     void displayCoreMembers() const {
      cout << "Events department coremembers by " << name << ": ";
      for (auto coremember : coremembers) {
          cout << coremember->getName() << " ";
      }
      cout << endl;
  }
}

void checkRole(string role, Event* registerEvent) {
    cout<<"Enter the following details :- "<<endl;
 if(role == "participant") {
   string participantID, name;
   cout<<"Enter Participant Id: "; cin>>participantID;
   cout<<"\nEnter Participand Name: "; cin>>name;
   Participant* registerParticipant = new Participant(participantID,
name);
   registerParticipant->registerForEvent(registerEvent);
   registerEvent->registerParticipant(registerParticipant);
 } else if(role == "coremember") {
   string name, spec; int id;
   cout<<"\nEnter name: "; cin>>name;
   cout<<"\nEnter spec: ";cin>>spec;
   cout<<"\nEnter id "; cin>>id;
   CoreMember* registerCoremember = new CoreMember(name,id,spec);
   registerEvent->registerEventCoremember(registerCoremember);
```

```
    department* departmentRegister = new department(spec);
    departmentRegister->registerMember(registerCoremember);
    //access the coordinators available
  } else if(role == "coordinator") {
    string name,id;
    Coordinator* registerCoordinator = new Coordinator(name,id);
  }else {
    int id; string name;
     cout<<"\nEnter the id: ";cin>>id;
     cout<<"\nEnter the name: ";cin>>name;
     Workforce* registerWorkforce = new Workforce(id,name);
     Coordinator* _coordinator = registerEvent->getCoordinator();
     registerWorkforce->assignCoordinator(_coordinator);
  }
}

int main() {
    cout<<"Register for the event"<<endl;
    cout<<"Enter eventName : "<<endl;
    string eventName; cin >> eventName;
    cout<<"Enter eventId : "<<endl;
    string eventId; cin>>eventId;
    Event* registerEvent = new Event(eventId,eventName);
    cout<<"Enter your role : "<<endl;
    string role; cin>>role;
    int size = role.length();
    for(int i = 0; i < size; i++) {
        role[i] = tolower(role[i]);
    }
    checkRole(role,registerEvent);
    return 0;
}
```

Coordinator (Muskan)

```
// Coordinator class
class Coordinator {
private:
   string name;
   CoreMember* coreMember;  // Coordinator reports to a Core Member
   vector<Workforce*> workforceMembers;  // Coordinator has workforce members
   vector<std::string> events;  // Coordinated events

public:
   Coordinator(const std::string& n, CoreMember* cm) : name(n), coreMember(cm) {}

   void addWorkforceMember(Workforce* wf) {
       workforceMembers.push_back(wf);
   }

   void addEvent(const std::string& event) {
       events.push_back(event);
   }

   void displayCoordinatorInfo() const {
       cout << "Coordinator: " << name << ", Reports to Core Member: " <<
coreMember->name << "\n";
       cout << "Coordinated Events: ";
       for (const std::string& event : events) {
           cout << event << ", ";
       }
       cout << "\nWorkforce Members: ";
       for (const Workforce* wf : workforceMembers) {
           cout << wf->getName() << ", ";
       }
       cout << "\n\n";
   }
};
```