You are working with the **DECS society** to organize an **annual picnic event**. The system should manage **budgeting, ticket sales, and attendee lists**.

**Requirements:**

1. Implement a **base class Event** to store the event **name, date, and venue**.
2. Implement a **derived class Picnic** that:

* Manages **fixed total tickets** (default 2000) and **total budget** (default 2,000,000).
* Keeps track of **tickets sold, remaining budget, and attendees (faculty & students)**.
* **Explicitly create methods** to **add faculty and students** to their respective lists.

1. Implement a **final derived class PicnicManagement** that:

* Allows selling tickets (1 or 2 at a time).
* Manages budget utilization (ensuring funds are not overdrawn).

1. **Use constructor chaining** to initialize attributes.
2. **Ensure total tickets and budget remain constant** once initialized.

#include <iostream>

using namespace std;

// Base Class: Event

class Event {

protected:

string eventName;

string eventDate;

string venue;

public:

// Parameterized Constructor

Event(string name, string date, string loc) : eventName(name), eventDate(date), venue(loc) {

cout << "Event Created: " << eventName << " on " << eventDate << " at " << venue << endl;

}

};

// Derived Class: Picnic (inherits from Event)

class Picnic : public Event {

protected:

const int totalTickets; // Fixed total number of tickets per instance

int ticketsSold;

const int totalBudget; // Fixed total budget per instance

int remainingBudget;

string faculty[10]; // Fixed size array for faculty names

string students[20]; // Fixed size array for student names

int facultyCount, studentCount;

public:

// Parameterized Constructor (Constructor Chaining)

Picnic(string name, string date, string loc, int tickets, int budget)

: Event(name, date, loc), totalTickets(2000), totalBudget(2000000), ticketsSold(0), remainingBudget(totalBudget),

facultyCount(0), studentCount(0) {

cout << "Picnic Initialized with " << totalTickets << " tickets and budget: " << totalBudget << endl;

}

// Function to add faculty members

void addFaculty(string name) {

if (facultyCount < 10) {

faculty[facultyCount] = name;

facultyCount++;

} else {

cout << "Faculty list is full!" << endl;

}

}

// Function to add students

void addStudent(string name) {

if (studentCount < 20) {

students[studentCount] = name;

studentCount++;

} else {

cout << "Student list is full!" << endl;

}

}

};

// Final Derived Class: PicnicManagement (inherits from Picnic)

class PicnicManagement : public Picnic {

public:

// Parameterized Constructor (Constructor Chaining)

PicnicManagement(string name, string date, string loc, int tickets, int budget)

: Picnic(name, date, loc, tickets, budget) {}

// Function to utilize funds

bool utilizeFunds(int amount, string reason) {

if (amount <= remainingBudget) {

remainingBudget -= amount;

cout << "Funds Utilized: " << amount << " for " << reason << ". Remaining Budget: " << remainingBudget << endl;

return true;

}

cout << "Insufficient funds for: " << reason << endl;

return false;

}

// Function to sell tickets

int sellTicket(int amount) {

if (amount < 1 || amount > 2) {

cout << "You can only buy 1 or 2 tickets at a time!" << endl;

return amount;

}

int availableTickets = totalTickets - ticketsSold;

int ticketsToSell = 0;

if (amount <= availableTickets) {

ticketsToSell = amount;

} else {

ticketsToSell = availableTickets;

}

ticketsSold += ticketsToSell;

remainingBudget += ticketsToSell \* 1500;

cout << "Tickets Sold: " << ticketsToSell << ". Remaining Tickets: " << (totalTickets - ticketsSold) << endl;

cout << "Updated Budget: " << remainingBudget << endl;

return amount - ticketsToSell; // Returning the number of tickets that couldn't be sold

}

};

// Main Function

int main() {

PicnicManagement picnic("DECS Annual Picnic", "March 24", "Beach Resort", 2000, 2000000);

picnic.sellTicket(2);

picnic.sellTicket(1);

picnic.utilizeFunds(500000, "Beach Booking");

picnic.sellTicket(5); // Invalid case

picnic.utilizeFunds(2000000, "Extra Lodges"); // Invalid case

return 0;

}