Course: Object Oriented Programming with C++

1. **Title: Design Activity document of Open Ended** **Assessment:**

Tournament Simulation of Various Games

# Team members list with roll numbers:

Srithan M S: 01FE21BCS016 (503)

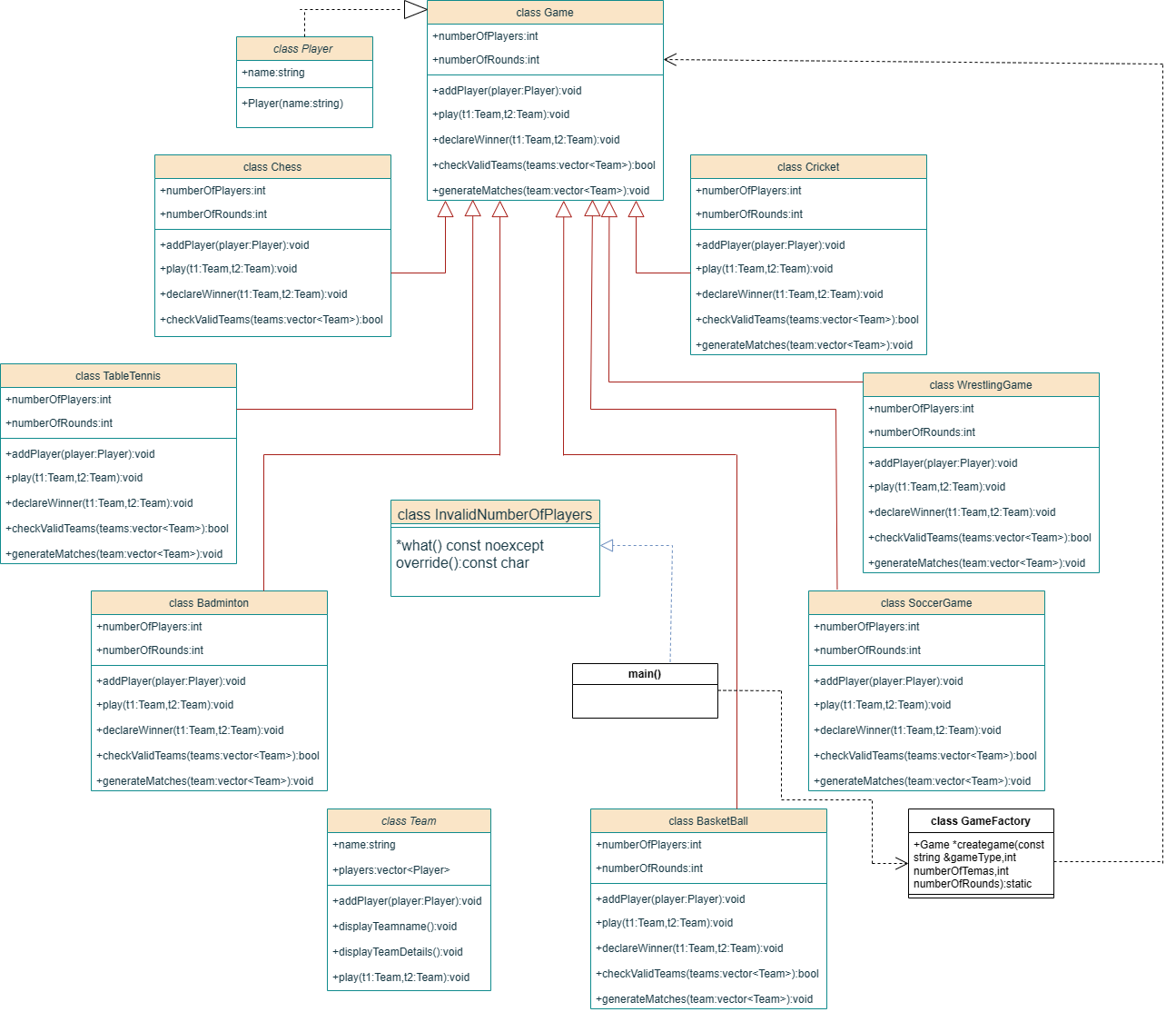
Sanjana Hubballi: 01FE21BCS022 (505)

# Problem Definition (Description):

It is inefficient and inconsistent how different games imitate competitions. We require a better software solution that streamlines the procedure by utilising basic programming ideas (OOPs). This system need to make it simple for organizers to set guidelines, plan games, keep score, rank players, and generate reports. We hope to develop a user-friendly system that can simulate tournaments for numerous games utilizing these programming techniques.

# 4. List of objects identified :

* Team
* Player
* Game
* Chess
* Soccer
* Basketball
* Cricket
* Wrestling
* Badminton
* TableTennis



# 6 . Description of each class:

**Player**: This class inherits from class Subject and shows the name of players.It has a default constructor and a function which prints the player details.

**HelpMenu**: This class stores the game instructions of each game. It has a default constructor and a function printMenu() which displays the game instructions for the particular game being played.

**Game**: This is an abstract class which contains the game details along with the player details and the number of Teams and number of Rounds for the game. This is a base class which is inherited by 7 other games.

**Chess**: This is a derived class inheriting from the base class Game. It has a parameterized

constructor. The function addPlayer(player:Player) adds a player to the Chess game. The function play(t1:Team,t2:Team) displays the matches between two teams.The declareWinner(t1:Team,t2:Team) displays the winner of the Chess game.The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team.

**Badminton**: This is a derived class inheriting from base class Game. It has a parameterized constructor. The function addPlayer(player:Player) adds a player to the Badminton game.The function play(t1:Team,t2:Team) displays the matches held between two teams. The declareWinner(t1:Team,t2:Team) displays the winner of the Badminton game. The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team. The function generateMatches(team<vector>Team) generates all possible matches between the teams and play the matches between the two teams.

**Cricket**: This is a derived class inheriting from base class Game.It has a parameterized constructor. The function addPlayer(player:Player) adds a player to the Cricket game. The function play(t1:Team,t2:Team) displays the matches held between two teams. The declareWinner(t1:Team,t2:Team) displays the winner of the Cricket game. The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team. The function generateMatches(team<vector>Team) generates all possible matches between the teams and play the matches between the two teams.

**Basketball:** This is a derived class inheriting from base class Game.It has a parameterized constructor. The function addPlayer(player:Player) adds a player to the Basketball game. The function play(t1:Team,t2:Team) displays the matches held between two teams. The declareWinner(t1:Team,t2:Team) displays the winner of the Basketball game. The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team. The function generateMatches(team<vector>Team) generates all possible matches between the teams and play the matches between the two teams.

**Wrestling:** This is a derived class inheriting from base class Game. It has a parameterized constructor. The function addPlayer(player:Player) adds a player to the Wrestling game. The function play(t1:Team,t2:Team) displays the matches held between two teams. The declareWinner(t1:Team,t2:Team) displays the winner of the Wrestling game. The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team. The function generateMatches(team<vector>Team) generates all possible matches between the teams and play the matches between the two teams.

**Soccer:** This is a derived class inheriting from base class Game. It has a parameterized constructor. The function addPlayer(player:Player) adds a player to the Soccer Game. The function play(t1:Team,t2:Team) adds a player to the Soccer Game. The function play(t1:Team,t2:Team) displays the matches held between two teams. The declareWinner(t1:Team,t2:Team) displays the winner of the game. The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team. The function generateMatches(team<vector>Team) generates all possible matches between the teams and play the matches between the two teams.

**TableTennis:** This is a derived class inheriting from base class Game.It has a parameterized constructor. The function addPlayer(player:Player) adds a player to the TableTennis game. The function play(t1:Team,t2:Team) displays the matches held between two teams. The declareWinner(t1:Team,t2:Team) displays the winner of the TableTennis game. The function checkValidTeam(teams<vector>Team) checks if there are two or fewer players from each team. The function generateMatches(team<vector>Team) generates all possible matches between the teams and play the matches between the two teams.

**InvalidNumberOfPlayers**: This class is used to handle all the errors generated for termination of the program.

# 7.Main function (Flow of object creation an function call)

The user enters the number of teams, number of rounds and selects a game type. Based on the selected game type, a specific game object is created using the GameFactory class. The user enters the names of teams and players for each team. The players are added to the respective team objects. The Game object’s checkValidTeam() method is called to verify the team’s validity based on the game’s rules. If the teams are valid, the generateMatches() method of the selected Game object is called. If the teams are valid, the generateMatches() method of the selected Game object is called. The generateMatches() method creates and executes matches between all possible pairs of teams for the specified number of rounds. During each match, the play() , method of the Team class is called, displaying the match details. The user enters the scores for the matches and the declareWinner() method of the Team class determines the match winner. After each match, the program proceeds to the next match until all matches are completed. The program prints the winner of each match and proceeds to the next round. Once all rounds are completed, the program finishes execution and memory is released by deleting the Game object.

int main() {

cout << "Welcome to the Game Management System!" << endl;

try {

int numberOfTeams, numberOfRounds;

cout << "Enter the number of teams: ";

cin >> numberOfTeams;

if (numberOfTeams <= 0) {

throw invalid\_argument("Number of teams must be greater than 0.");

}

cout << "Enter the number of rounds: ";

cin >> numberOfRounds;

if (numberOfRounds <= 0) {

throw invalid\_argument("Number of rounds must be greater than 0.");

}

string gameType;

cout << "Choose a game: Chess, Cricket, Wrestling, Badminton, Table "

"Tennis, Basketball, or Soccer: ";

cin >> gameType;

Game\*game =GameFactory::createGame(gameType, numberOfTeams, numberOfRounds);

vector<Team> teams;

for (int i = 1; i <= numberOfTeams; i++) {

Team team;

cout << "Enter the name of Team " << i << ": ";

cin >> team.name;

int numberofPlayers;

cout << "Enter the number of players in Team " << i << ": ";

cin >> numberofPlayers;

if (numberofPlayers <= 0)

{

throw InvalidNumberOfPlayers();

}

if (gameType == "Chess")

{

if(numberofPlayers!=2)

{throw InvalidNumberOfPlayers();}

} else if (gameType == "Cricket")

{

if(numberofPlayers != 11)

{throw InvalidNumberOfPlayers();}

} else if (gameType == "Wrestling")

{

if( numberofPlayers!=1)

{throw InvalidNumberOfPlayers();}

} else if (gameType == "Badminton")

{

if(numberofPlayers>4)

{throw InvalidNumberOfPlayers();}

} else if (gameType == "TableTennis")

{

if(numberofPlayers>4)

{throw InvalidNumberOfPlayers();}

}

else if (gameType == "Basketball")

{

if(numberofPlayers!=12)

{

throw InvalidNumberOfPlayers();

}

}

else if (gameType == "Soccer")

{

if( numberofPlayers!= 11)

{ throw InvalidNumberOfPlayers();

}

}

for (int j = 1; j <= numberofPlayers; j++) {

string playerName;

cout << "Enter the name of Player " << j << " in Team " << i << ": ";

cin >> playerName;

Player player(playerName);

team.addPlayer(player);

}

teams.push\_back(team);

}

if (!game->checkValidTeams(teams)) {

throw invalid\_argument("Invalid team configuration.");

} else {

game->generateMatches(teams);

}

delete game;

}

catch (const InvalidNumberOfPlayers& e)

{

cerr << "Error: " << e.what() << endl;

}

catch (const exception &e) {

cerr << "Error: " << e.what() << endl;

} catch (...) {

cerr << "Unknown error occurred." << endl;

}

return 0;

}

# 8.Use of Standard Design Patterns :

Factory design pattern is used for the above application.

**Factory design pattern:**

This design pattern is used to choose any of the game from the 7 types of game from the user input where class Game acts as the client which uses the class Game to create the object upon the request from the player input given in main() function.

