Lecture# 12 outline and homework

Today's Content:

- **Composition and aggregation**

```
void printall(void);///A gloabal function which we make friend of player class

class Date {
	int year;
	int month;
	int day;

public:
	Date(int y = 1, int m = 1, int d = 1)
	{
		cout << "\nIn date constructor";
		year = y;
		month = m;
		day = d;
	}
	void printdate()
	{
		cout <<"\nDate: "<< year << ":" << month << ":" << day << endl;
	}

};


class player
{
	int Id;//               //> .
	char name;//                //> .
	int size;//                     //>  non-static and  non-constant data members
	int *Scores;//              //> .
	float Average;//        //> .

	static int count;  //static data members

	const char gender;//Constant data member

	//Date DoB; ///composition
	Date *DoB; ///composition

	//Date &DoM;///aggregation

	Date *DoM;///aggregation

public:
	player(Date *, int =1, int = 1, int= 1, int = 0, char = 'a', int s = 2, char =
'M', int * = NULL);//Default parameterized constructor
```

```
	//player(int);

	/*player(double a):gender('M')///conversion function using constructor
	{
		Average = a;
	}*/



	//player();//default constructor discuss during lecture

	////Copy constructor//discuss during lecture
	player(const player&);

	// .......  Utility Functions ........
	player& calAverage(void);
	player& print(void);

	//......  Setter or Mutator Functions ......
	void setId(int);
	void setName(char);
	void setsize(int);
	void setScores(int *);//interesting

	// ..... Accessor or Getter functions .......
	int getID(void) const;
	char getName(void) const;
	float getAverage(void);
	int getsize(void) const;
	//How to write getscores function ?????

	static void showcount() // static function
	{
		//cout << name;
		cout << "\nValue of count" << count;
	}

	~player(); //Destructor

	///operator overloading
	void operator=(const player &);

	///implement here other arithmetic operators like operator-, operator*, operator/,
operator%, operator--
	/*player& operator+(player &);
	player& operator++(void);//prefix ++
	player& operator++(int);//postfix ++

	void operator|(const player &p)
	{
		cout << this->size<<endl;
		cout << p.size<<endl;

	}*/

	int& operator[](int i)
	{
		return Scores[i];
```

```
        }

        //Implement here other comparison and logical operators
        bool operator<(player &);

        //conversion function

        operator int()
        {
                return Id;
        }


        /////friend functions
        friend ostream& operator<<(ostream& out, player &p);
        friend void printall();//Granting printall() function as friend of class player

};
//////////////////////// . . . define class functions out of line/scope . . . ////////

int player::count = 0;//assigning value to static data member of class

player::player(Date *dm, int y, int m , int d, int i, char n, int s, char g, int *arr) :
Id(i), name(n), size(s), gender(g)//Constant data member must need intilizer with
constructor
{
        DoM = dm;

        cout << "\nInside parameterized Constructor : \n";

        DoB = new Date ( y,m,d );

        if (arr == NULL)
        {

                Scores = new int[size];
                cout << "Enter values of " << size << "  player : ";

                for (int i = 0; i < size; i++)
                {
                        cin >> Scores[i];
                }

        }
        else
        {
                Scores = new int[size];
                cout << "Enter values of   " << size << " students : ";
                for (int i = 0; i < size; i++)
                {
                        cout << "\nEnter " << i + 1 << " Value : ";
                        cin >> Scores[i];
                }

        }
        player::calAverage();//calculating average in constructor
        count++;
}
```

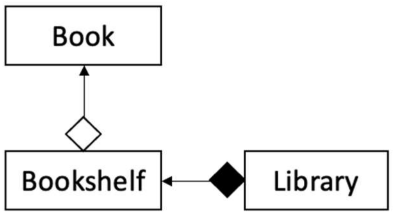
```
/*player::player():gender('M')//commented it as default parameterized
{

}*/
///Defination of copy constructor
////Copy constructor
player::player(const player & p) :gender(p.gender)
{
	cout << "\nIn Copy Constructor\n";
	this->Id = p.Id;
	this->size = p.size;

	this->Scores = new int[this->size];
	cout << "\nEnter " << size << " Values for scores";
	for (int i = 0; i < this->size; i++)
	{
		cout << "\nEnter " << i + 1 << " Value : ";
		cin >> this->Scores[i];
	}
	player::calAverage();
	count++;
}



// .......  Utility Functions ........

player& player::calAverage(void)
{

	cout << "\nInside CalculateAverage() Function\n";
	int s = 0;
	for (int i = 0; i < size; i++)
	{
		s += Scores[i];
	}
	this->Average = float(s) / size;

	return *this;
}
player& player::print()
{
	DoB->printdate();
	cout << "\n................................";
	cout << "\nInside print() function";
	cout << "\nName of player is :   " << name;
	cout << "\nID of player is :   " << Id;
	cout << "\nTotal matches played are : " << size;
	cout << "\nScores of player is:   ";
	for (int i = 0; i < size; i++)
	{
		cout << Scores[i] << " ";
	}
	cout << endl;

	cout << "\nAverage of player is:   " << this->Average;
	cout << "\n...............................\n";
	return *this;
}

//......  Setter or Mutator Functions ......
void player::setId(int i)
{
	cout << "\nInside setId() function";
	Id = i;
}

void player::setName(char c)
{
	cout << "\nInside setName() function";
	//name = c;

}

void player::setsize(int s)
{

	cout << "\nInside setsize() function";
	this->size = s;
}
void player::setScores(int *arr)
{
	cout << "\nInside setScores() function";
	delete[] Scores;
	Scores = NULL;
	Scores = new int[size];

	for (int i = 0; i < size; i++)
	{
		Scores[i] = arr[i];
	}


}
// ..... Accessor or Getter functions .......
int player::getID(void) const
{
	cout << "\nInside getId() function\n";
	return this->Id;
}
char player::getName(void) const
{
	cout << "\nInside getName() function\n";
	return name;
}
float player::getAverage()///an interesting fact inside function
{
	cout << "\nInside getAverage() function\n";
	player::calAverage();
	return Average;

}
int player::getsize(void) const
{
        return (*this).size;//return this->size;
}

//Definition of Destructor
player::~player() //Destructor
{
        cout << "\nInside Destructor that Delete Dynamic Memory\n";

        delete[] Scores;
        delete DoB;
        count--;
}
void player::operator=(const player &p)
{
        this->Id = p.Id;
        this->size = p.size;
        delete[] Scores;
        this->Scores = new int[size];
        cout << "\nEnter " << size << " Values for scores";
        for (int i = 0; i < this->size; i++)
        {
                cout << "\nEnter " << i + 1 << " Value : ";
                cin >> this->Scores[i];
        }
        player::calAverage();
}
/////////////////////////////////////////////////operator
overloading//////////////////////////////
/*player& player::operator+(player &p)
{
        player pt;

        pt.Average = this->Average + p.Average;

        return pt;
}

bool player::operator<(player &p)
{
        if (this->Average < p.Average)
                return true;
        else
                return false;

}

player& player::operator++(void)
{
        cout << "\ninside operator++(void) function ";
        for (int i = 0; i < this->size; i++)
                Scores[i]++;
        return *this;

}

player& player::operator++(int)
{
        cout << "\ninside operator++(int) function ";
```

```
	for (int i = 0; i < this->size; i++)
		Scores[i]++;

	return *this;

}*/

///.......Defination of printall() global function ..........//
//Read this function carefully and implement it in main ...............///
void printall()
{
	cout << "\n......Inside printall() global function.......\n";
	Date *pt = new Date{ 2,2,2 };
	player p(pt, 2, 7, 11, 3, 'A', 4);
	cout << "\n................................";
	cout << "\nInside print() function";
	cout << "\nName of player is :   " << p.name;
	cout << "\nID of player is :   " << p.Id;
	cout << "\nTotal matches played are : " << p.size;
	cout << "\nScores of player is:   ";
	for (int i = 0; i < p.size; i++)
	{
		cout << p.Scores[i] << " ";
	}
	cout << endl;

	cout << "\nAverage of player is:   " << p.Average;
	cout << "\n................................\n";
}


///implementation of operator<<()
ostream& operator<<(ostream& out, player &p)
{
	out << "\n................................";
	out << "\nInside operator<<() function";
	out << "\nName of player is :   " << p.name;
	out << "\nID of player is :   " << p.Id;
	out << "\nTotal matches played are : " << p.size;
	out << "\nScores of player is:   ";
	for (int i = 0; i < p.size; i++)
	{
		out << p.Scores[i] << " ";
	}
	out << endl;

	out << "\nAverage of player is:   " << p.Average;
	out << "\n................................\n";
	return out;

}

int main()
{
	//Date o1(23, 7, 17);
	// write your implementstion code here

	Date *p = new Date{ 2,2,2 };
```

```
        player p1(p, 2,7,11, 3, 'A', 4) , p2(p);

        p1.print();

        delete p;

}
```

Problem:

Label the relationship shown below. Then write code to fully demonstrate the relationships between the classes shown, complete with a driver program that shows how objects will be instantiated.


Book
Bookshelf
Library