***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B10***

***Software Development Lab – II [15B17CI271]***

***Assignment Sheet***

***Week 8***

1. *Multiple students can associate with a single Department and single student can associate with multiple Departments, but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.*

*WAP in C++ to model the relationships.*

**

***Solution:***

#include<iostream.h>

class Student;

class Department

{

 char\* name\_p;

 public:

 Department(char \*dName)

 {

 cout<<"Department::ctor\n";

 name\_p = new char(sizeof(strlen(dName)));  name\_p = dName;

 }

 char\* dName() const

 {

 return(name\_p);

 }

 ~Department()

 {

 cout<<"Department::dtor\n";

 delete(name\_p);

 }

};

class Student

{

 char\* name\_p;

 public:

 Student(char \*sName)

 {

 cout<<"Student::ctor\n";

 name\_p = new char(sizeof(strlen(sName)));  name\_p = sName;

 }

 char\* sName()const

 {

 return(name\_p);

 }

 ~Student()

 {

 cout<<"Student::dtor\n";

 delete(name\_p);

 };

};

class Course

{

 Student \* std\_p;

 Department \* dept\_p;

 char \* courseName\_p;

 static unsigned int index;

 static Course \*courseList[4];

 public:

 Course(char\* crseName, Student\* student, Department\* dept):  courseName\_p(0), std\_p(student), dept\_p(dept)  {

 cout<<"Course:ctor\n";

 if (index < 4)

 {

 courseName\_p = new char(sizeof(strlen(crseName)));  courseName\_p = crseName;

 //insert this Course in courseList

 courseList[index] = this;

 ++index;

 }

 else

 {

 cout<<"Cannot accomodate any more Course\n";  }

 };

 ~Course()

 {

 cout<<"Course:dtor\n";

 delete (courseName\_p);

 };

 static char\* findStudent(char \*crseName, char\* deptName)  {

 for(int i=0; i<index; i++)

 {

 if ( (courseList[i]->getCourseName() == crseName) &&   (courseList[i]->getDeptName() == deptName) )  {

 return(courseList[i]->getStdName());  }

 }

 }

 char \* getStdName()const {return(std\_p->sName());};  char \* getDeptName() const {return(dept\_p->dName());};  char \* getCourseName()const {return(courseName\_p);}; };

unsigned int Course::index =0;

Course\* Course::courseList[4] = {0,0,0,0};

int main()

{

 int i;

 cout<<"\nExample of Association class...\n";  cout<<"-----------------------------------\n\n";

 cout<<"We have got 4 students ...\n";

 Student \*studentNames[4] = {new Student("Meera"), new  Student("Moina"), new Student("Teena"), new Student("Mridula")} ;

 cout<<"\n";

 cout<<"We have got 2 Departments...\n";

 Department \*departNames[2] = {new Department("Mathemetics"), new Department("ComputerSceince")} ;

 cout<<"\n";

 cout<<"Here class Course Associates Student and Department, with a Course name ...\n";

 Course course1("DataStructure",studentNames[0], departNames[1]); Course course2("Maths",studentNames[3], departNames[0]);  Course course3("Geometry",studentNames[2], departNames[0]);  Course course4("CA",studentNames[1], departNames[1]);

 cout<<"\n";

 cout<<"Finding a Student using Course and Department...\n";  cout<<"Student who has taken Maths Course in Mathemetics Department is:"<<Course::findStudent("Maths", "Mathemetics")<<endl;

 cout<<"\n";

 cout<<"Deletion of objects...\n\n";

 for(i=0; i<4; ++i)

 {

 delete studentNames[i];

 }

 cout<<"\n";

 for(i=0; i<2; ++i)

 {

 delete departNames[i];

 }

 cout<<"\n";

 return(0);

}

**output:**

------

Example of Association class...

-----------------------------------

We have got 4 students ...

Student::ctor

Student::ctor

Student::ctor

Student::ctor

We have got 2 Departments...

Department::ctor

Department::ctor

Here class Course Associates Student and Department, with a Course name ...

Course:ctor

Course:ctor

Course:ctor

Course:ctor

Finding a Student using Course and Department...

Student who has taken Maths Course in Mathemetics Department  is:Mridula

Deletion of objects...

Student::dtor

Student::dtor

Student::dtor

Student::dtor

Department::dtor

Department::dtor

Course:dtor

Course:dtor

Course:dtor

Course:dtor

1. *A single Employee can not belong to multiple Companies (legally!!), but if we delete the Company, Employee object will not destroy.*

*WAP in C++ to model the relationships.*

**

***Solution:***

#include<iostream.h>

class Employee

{

 public:

 Employee(char \*name){

 cout<<"Employee::ctor\n";

 myName\_p = new char(sizeof(strlen(name)));  myName\_p = name;

 }

 char\* disp(){return(myName\_p);};

 ~Employee()

 {

 cout<<"Employee:dtor\n\n";

 delete (myName\_p);

 }

 private:

 char \*myName\_p;

};

class Company

{

 public:

 Company(char \* compName, Employee\* emp){  cout<<"Company::ctor\n";

 name = new char(sizeof(strlen(compName)));   name = compName;

 myEmp\_p = emp;

 };

 ~Company()

 {

 cout<<"Company:dtor\n\n";

 myEmp\_p = NULL;

 };

 private:

 char \*name;

 Employee \*myEmp\_p;

};

int main()

{

 cout<<"\nExample of Aggregation Relationship \n";  cout<<"-----------------------------------------\n\n";

 {

 cout<<"Here, an Employee-Keerti works for Company-MS \n";  Employee emp("Keerti");

 {

 Company comp("MS", &emp);

 } // here Company object will be deleted, whereas Employee object is still there

 cout<<"At this point Company gets deleted...\n";  cout<<"\nBut Employee-"<<emp.disp();

 cout<<" is still there\n";

 } //here Employee object will be deleted

 return(0);

}

**output:**

-------

Example of Aggregation Relationship

-----------------------------------------

Here, an Employee-Keerti works for Company-MS  Employee::ctor

Company::ctor

Company:dtor

At this point Company gets deleted...

But Employee-Keerti is still there

Employee:dtor

***3.*** *House can contain multiple rooms. There is no independent life for room and any room can not belong to two different houses. If we delete the house, room will also be automatically deleted.*

*WAP in C++ to model the relationships.*

**

***Solution:***

#include<iostream.h>

class House;

class Room

{

 public:

 Room()

 {

 };

 static void createRoom\_v(Room\* (&room), House\* hse, char\* name) {

 room = new Room(hse, name);

 }

 Room(House\* hse, char\* myName)

 {

 cout<<"Room::ctor\n";

 myHse\_p = hse;

 if(NULL != myHse\_p)

 {

 name\_p = new char(sizeof(strlen(myName)));  name\_p = myName;

 }

 else

 {

 cout<<"Oops House itself is not Created Yet ...\n";  }

 };

 ~Room()

 {

 cout<<"Room:dtor\n";

 myHse\_p = NULL;

 delete (name\_p);

 };

 void disp()

 {

 cout<< name\_p;

 cout<<"\n";

 }

 static void initList\_v(Room \*(& roomsList\_p)[3])  {

 roomsList\_p[3] = new Room[3];

 }

 private:

 House \* myHse\_p;

 char \* name\_p;

};

class House

{

 public:

 House(char \*myName)

 {

 cout<<"House::ctor\n";

 name\_p = new char(sizeof(strlen(myName)));;  name\_p = myName;

 Room::initList\_v(roomsList\_p);

 Room\* myRoom;

 Room::createRoom\_v(myRoom, this, "Kitchen");  roomsList\_p[0] = myRoom;

 Room::createRoom\_v(myRoom, this, "BedRoom");  roomsList\_p[1] = myRoom;

 Room::createRoom\_v(myRoom, this, "Drwaing Room");  roomsList\_p[2] = myRoom;

 }

 ~House()

 {

 cout<<"House:dtor\n";

 unsigned int i;

 cout<<"Delete all the Rooms ...\n";  for(i=0; i<3; ++i)

 {

 if(roomsList\_p[i] != NULL)

 {

 delete (roomsList\_p[i]);

 }

 }

 delete [] roomsList\_p;

 delete (name\_p);

 }

 void disp()

 {

 cout<<"\n\nName of the House :"<<name\_p;

 if(roomsList\_p != NULL)

 {

 unsigned int i;

 cout<<"\n\nRooms details...\n";  for(i=0; i<3; ++i)

 {

 if(NULL != roomsList\_p[i])

 {

 roomsList\_p[i]->disp();

 }

 }

 cout<<"\n\n";

 }

 }

 private:

 char\* name\_p;

 Room\* roomsList\_p[3];

};

int main()

{

 cout<<"\nExample of Composition Relationship\n";  cout<<"-----------------------------------------\n\n";  House hse("Vishranti Nilaya");

 cout<<"\n\nHouse details...\n";

 hse.disp();

 cout<<"Here House itself creates the Rooms and Deletes as well, before it gets deletd...\n";

 return(0);

}

**output:**

-------

Example of Composition Relationship

-----------------------------------------

House::ctor

Room::ctor

Room::ctor

Room::ctor

House details...

Name of the House :Vishranti Nilaya

Rooms details...

Kitchen

BedRoom

Drwaing Room

Here House itself creates the Rooms and Deletes as well, before it gets deletd...

House:dtor

Delete all the Rooms ...

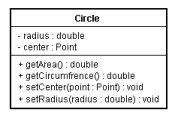
Room:dtor

Room:dtor

Room:dtor

***4.******Solution:***

*Class*

**

*class Circle {*

*private:*

*double radius;*

*Point center;*

*public:*

*setRadius(double radius);*

*setCenter(Point center);*

*double getArea();*

*double getCircumfrence();*

*};*

***5. Solution:***

*Aggregation*

**

*class Window*

*{*

*public:*

*//...*

*private:*

*vector itsShapes;*

*};*

***6. Solution:***

*#include <iostream>*

*using namespace std;*

*class Vehicle*

*{*

*public:*

*float wheelbase, power , seat;*

*void accelerator(){....}*

*void wheel(){....}*

*void brake(){....}*

*};*

*class Car : public Vehicle*

*{*

*public:*

*float height;*

*void getHeight()*

*{*

*cout << "enter value of height: "; cin >> power; }*

*};*

*class Lorry : public Vehicle*

*{*

*public:*

*void area()*

*{*

*cout << "area = " << seat\*2;*

*}*

*};*

*int main()*

*{*

*Car obj1;*

*Lorry obj2;*

*obj1.accelerator();*

*obj1.getHeight();*

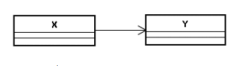
*obj2.area();*

*return 0;*

*} //end of program*

***7. Solution:***

*Association*

**

*Class X {*

*X(Y \*y) : y\_ptr(y) {}  void SetY(Y \*y) { y\_ptr = y; }*

*void f() { y\_ptr->Foo();}  ----*

*Y \*y\_ptr; // pointer };*