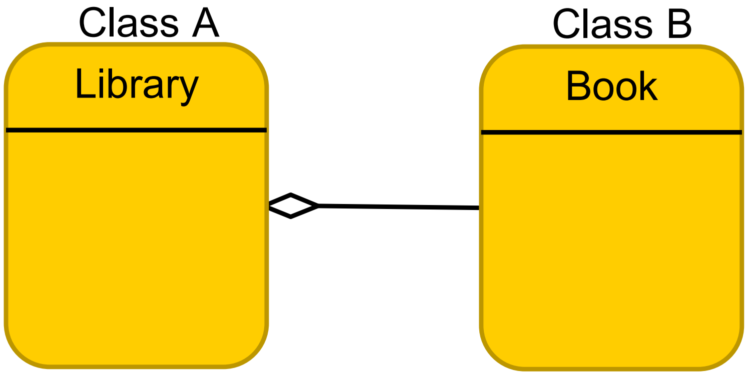
Class Relationship (Composition ,Association)

Class Relationship

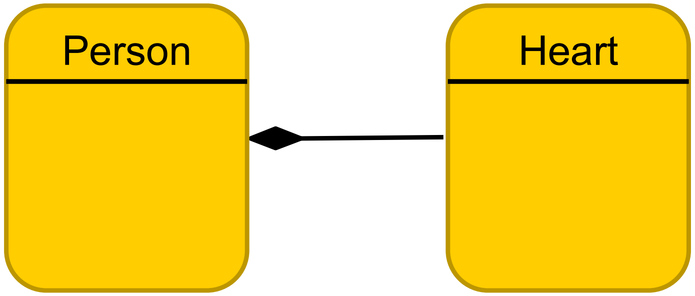
- Aggregation
  - Object within class as a member
    - Achieved through pointer or reference
  - *Has-a* relationship ,
  - *Whole-part* relationship
  - The whole not responsible for creating the part
  - The part could belong to more than one whole at a time
  - The part does not know about the existence of whole
    - Ex: Person – Address
    - Ex: library-book
    - Ex: Department-Employee
  - Creation of the part objects outside the whole class



```
Class B
{

}
class A
{
  B* objB;
};
```

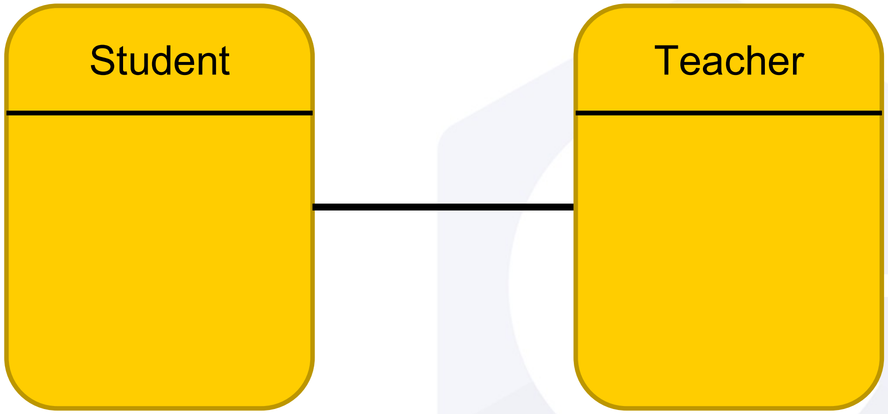
Class Relationship



- Composition
  - A stronger form of aggregation
  - *Has-a* relationship
  - *Death* relation
    - Kill the whole kill the part
  - The whole responsible for creating the parts
  - The part could belong to only one whole at a time
  - The part does not know about the existence of whole
    - Ex: Person-leg ,hand , head
    - Ex: website - webpage
    - Ex: Circle- Center (point)
  - Creation of the part objects inside the whole class

Class Relationship

- Association
  - Two classes communicate with each other
  - No ownership
  - Ex :Student – teacher
  - Ex: Patient – Doctor
  - Ex: Driver –Car



Using Graphics WinBGI

- Add *graphics.h* , *WinBGI.lib* files to project Directory
- #include graphics.h
- Project properties→configuration properties →linker→ Input→ edit (add winBGI.Lib)

Using Graphics WinBGI

- Graphics Methods

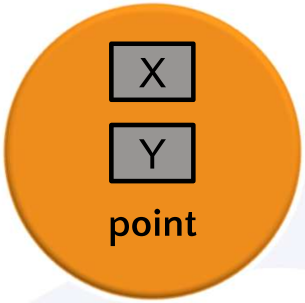
Borland Graphics Interface (BGI) Documentation (colorado.edu)

```
initwindow(1000, 1000, "Composition Example");
cleardevice();
closegraph();
Kbhit();
void circle(int x, int y, int radius);
void rectangle (int left, int top, int right, int bottom);
void line (int x1, int y1, int x2, int y2);
void setcolor (int color);
void setfillstyle(int pattern, int color);
void floodfill(int x, int y, int border);
```

Name	Value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

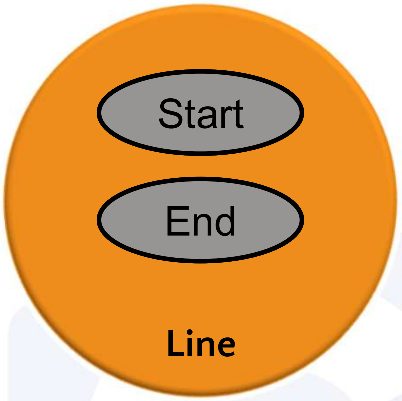
Composition Example

- Class point
  - X (int)
  - Y (int)



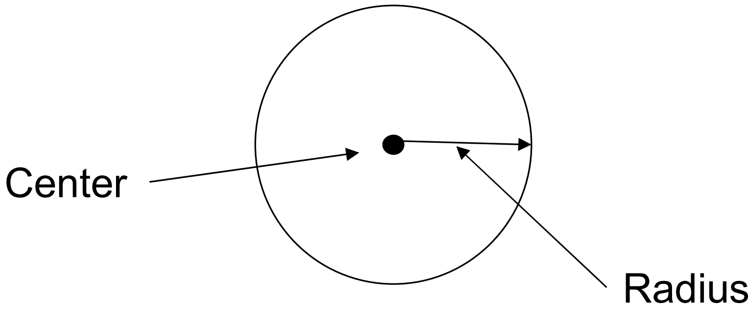
Composition Example

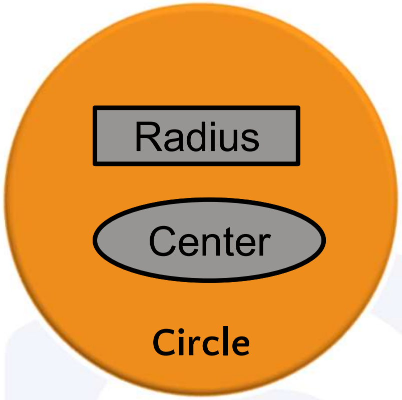
- Class Line
  - Start (point)
  - End (point)



Composition Example

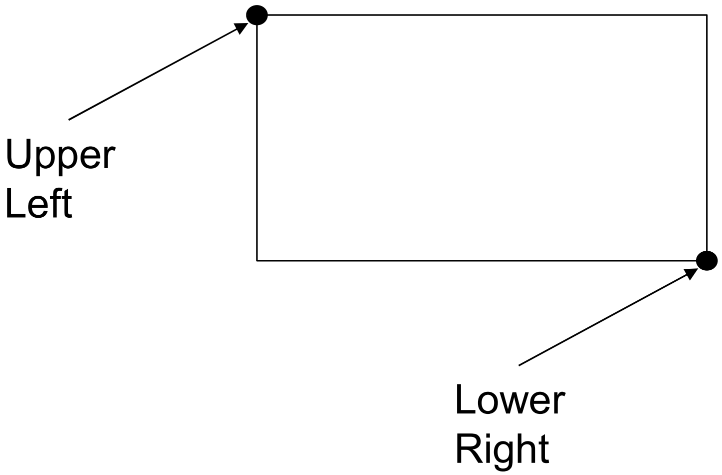
- Class circle
  - Radius (int)
  - Center (point)

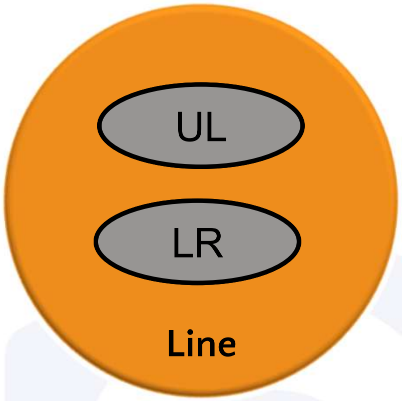




Composition Example

- Class Rect
  - UL (point)
  - LR (point)



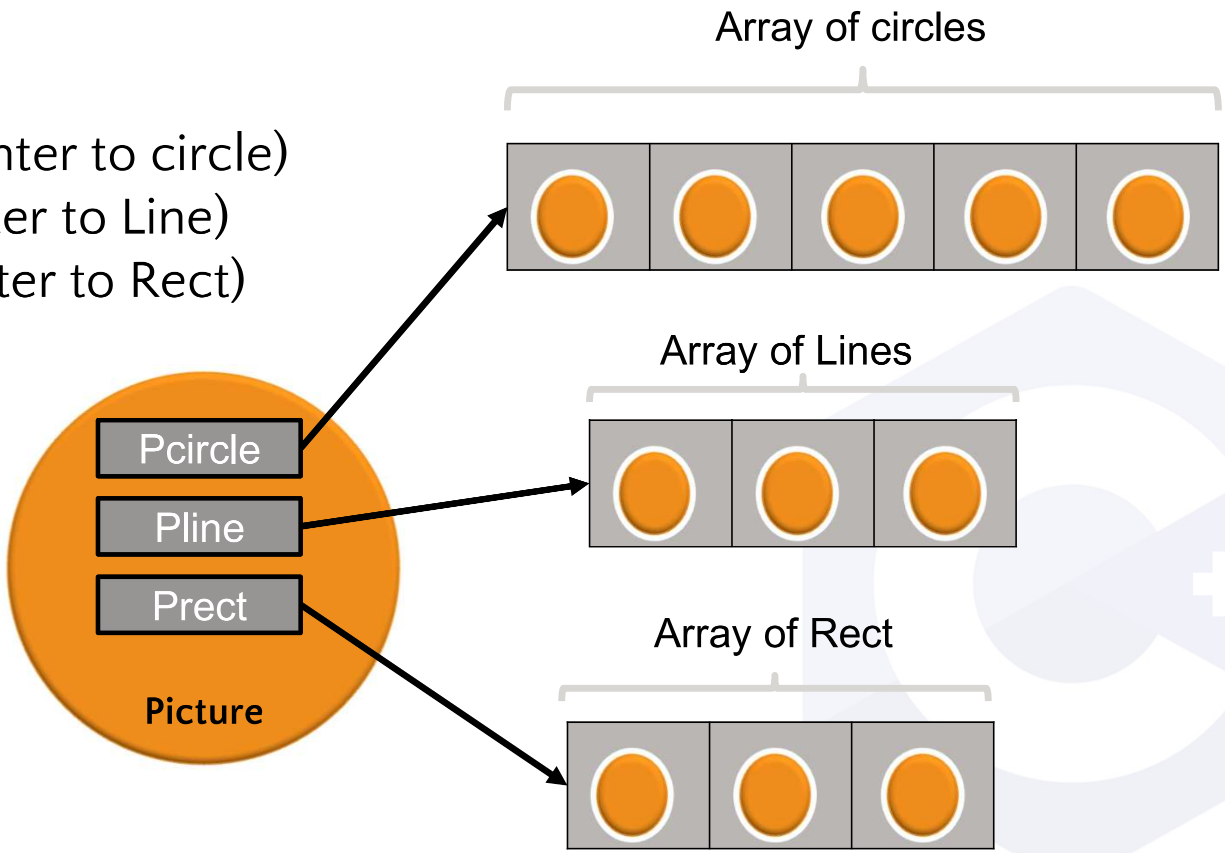


Assignment

- Design a program for draw Circle(s) , Line(s) ,Rectangle(s) using explained classes and functions

Aggregation Example

- Class Picture
  - Pcircle (pointer to circle)
  - Pline (pointer to Line)
  - Prect (pointer to Rect)



Aggregation Example

```
class picture
{
  private:
   Circle *pcircle;
    Rect *prect;
    Line *pline;
    int cnum;
    int rnum;
    int lnum;
  public:
    picture()
    {
     pcircle = NULL;
     prect = NULL;
      pline = NULL;
     cnum = rnum = lnum = 0;
    }
```

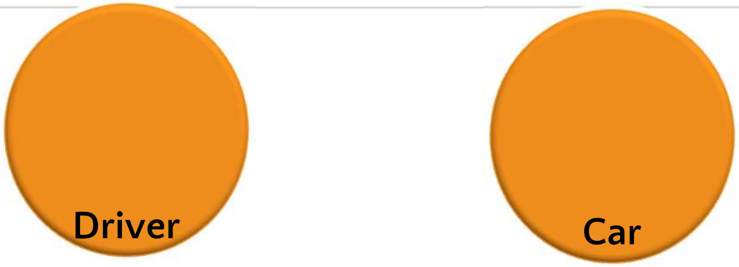
```
picture(Circle *c, Rect *r, Line *l,
            int cn, int rn, int ln)
    {
        pcircle = c;
        prect = r;
        pline = l;
        cnum = cn;
        rnum = rn;
        lnum = ln;
    }
```

Aggregation Example

```
void paint()
    {
        int i;
        cleardevice();
        for (i = 0; i < cnum; i++)
        {
            pcircle[i].draw();
        }
        for(i=0;i<rnum;i++)
        {
            prect[i].draw();
        }
        for(i=0;i<lnum;i++)
        {
            pline[i].draw();
        }
    }
```

Association Example

```
class Car
{
    private:
    char Model[30];
    int Year;
    public:
    Car()
    {   Model[0] = '\0';
        Year = 0;
    }
    Car(char* model_name,int year)
    {   strcpy(Model,model_name);
        Year = year;
    }
    void Move()
    {  cout << "Car :"<<Model<<"
       moving..." << endl;
    }
};
```



Driver

Car

Association Example

```
class Driver
{
    private:
    char Name[30];
    public:
    Driver()
    { Name[0] = '\0';
    }
    Driver(char* name)
    { strcpy(Name,name);
    }
    void Drive(Car c)
    {
        cout<<"driver:"<< Name<<"  drive ";
        c.Move();
    }
};
```

```
int main()
{
    Car c1("BMW", 2020);
    Car c2("Mercedes benz", 2020);

    Driver d1("Ahmed");
    d1.Drive(c1);
    d1.Drive(c2);
    system("pause");
    return 0;
}
```

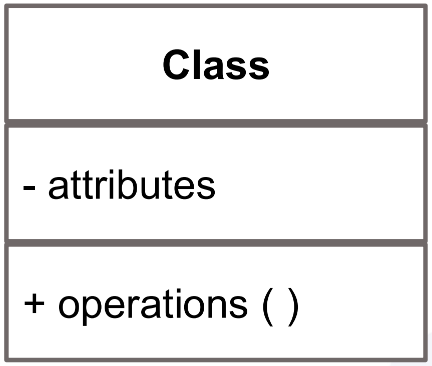
UML
Unified Modeling Language

UML

- A universally accepted way of describing software in diagrammatic form

Class

- Access modifiers
  - + (public)
  - (private)
  - # (protected)

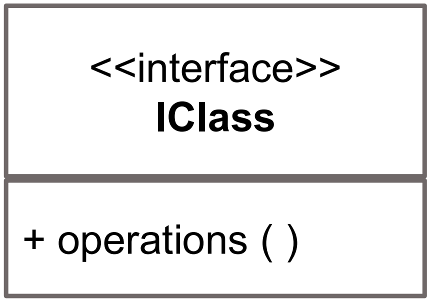


Class

- attributes

+ operations ( )

Interface or abstract Classes

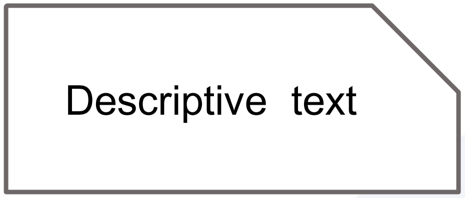


<<interface>>
IClass

+ operations ( )

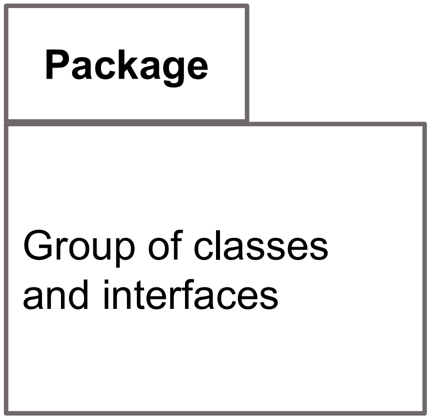
Note

- Description when needed



Descriptive text

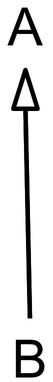
Package



Group of classes and interfaces

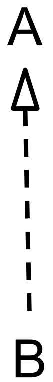
Inheritance

- B inherits from A



Realization

- B implements A



Association

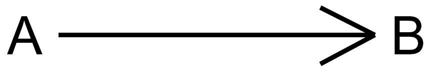
- A and B call and access each Other elements

A

B

Association (one way)

- A Can Call and Access B elements but not vise versa
- Example
  - Driver (A)   Car (B)

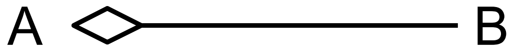


A

B

Aggregation

- A has a B , and B can Outlive A



Composition

- A has a B, B depends on A