Assignment 02
Hamim Hamid, 160041047
3 December, 2021
Design Pattern, CSE: 4851

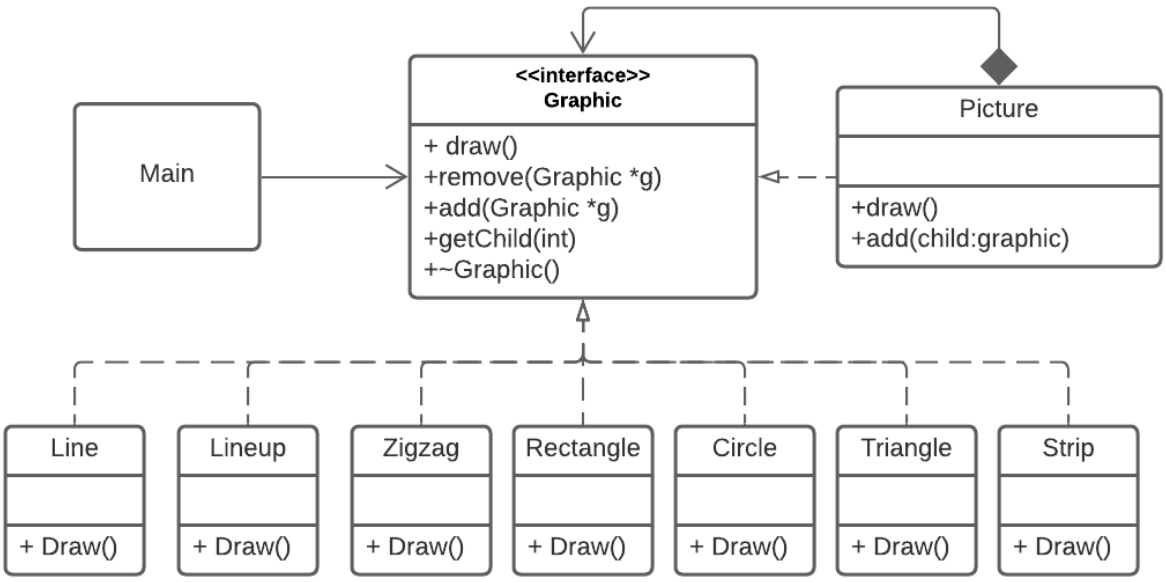
Age of Villagers

Objective: Using simple graphics shapes to create composite graphical features

Method: Implementing composite pattern and creating composite objects like house, river etc. with basic primary shapes like lines, triangles, circles etc. Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. It is a structural pattern. This pattern creates a class that contains group of its own objects. This class provides ways to modify its group of same objects.

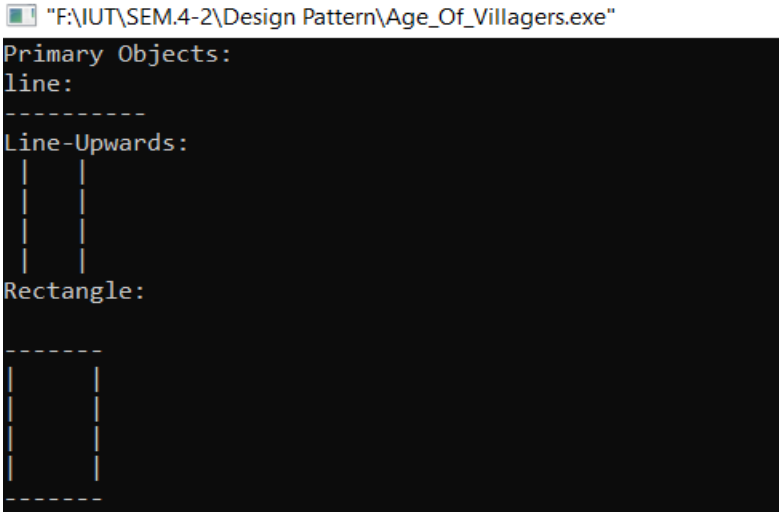
Medium: CPP as OOP language

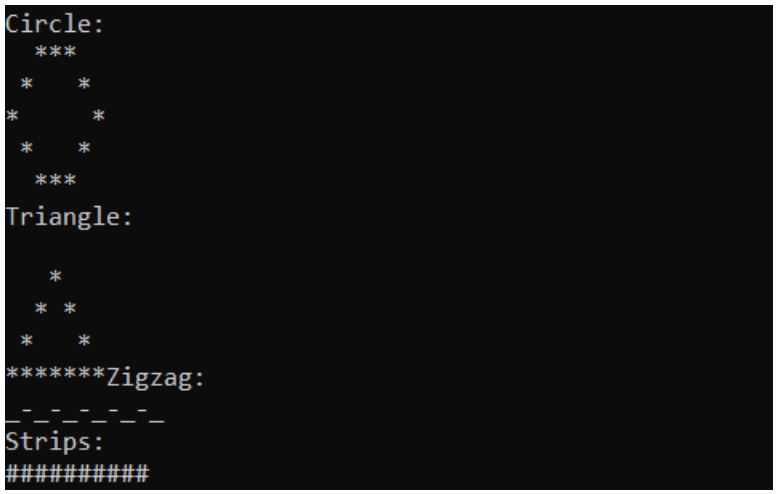
UML Class Diagram:



Graphical Result:

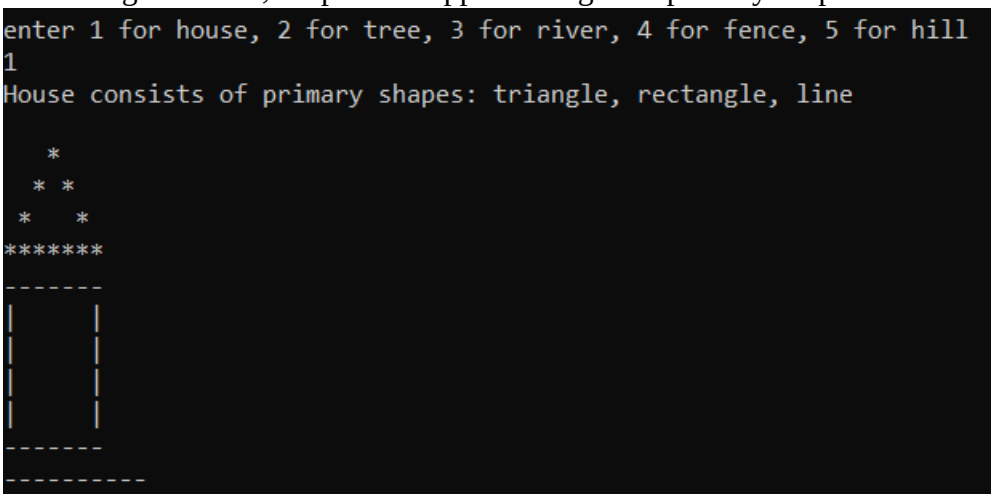
1. Upon entering gamer will get to know the basic graphical shapes that are: Line, Lineup, Zigzag, Rectangle, Circle, Triangle, Strip.

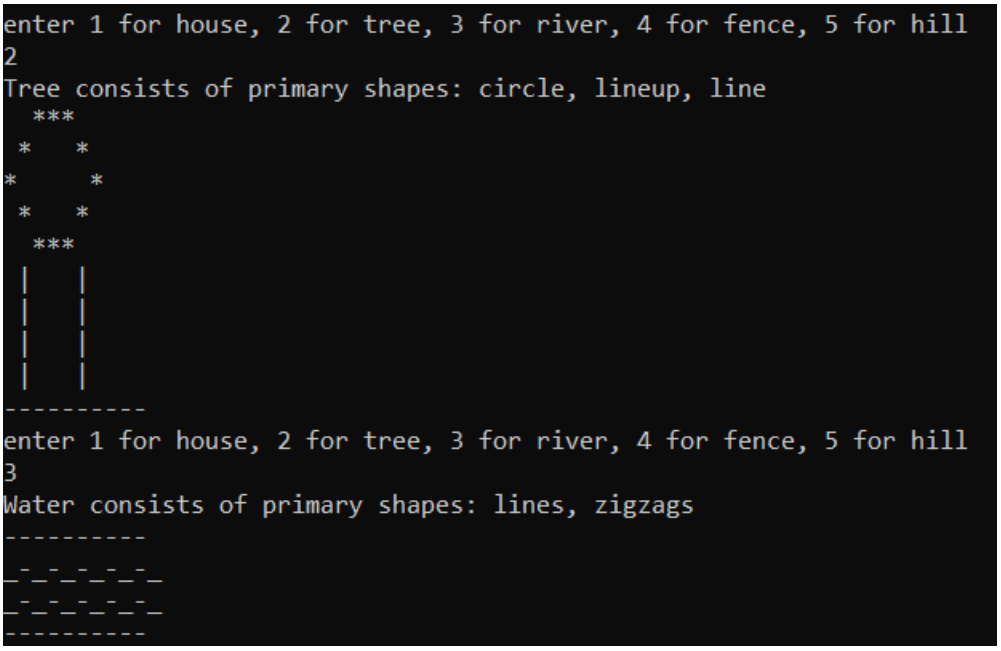




2. After seeing the gamer will get to choose the composite shapes, he/she wants:

3. According to choice, shapes will appear along with primary shape information:





```
enter 1 for house, 2 for tree, 3 for river, 4 for fence, 5 for hill
4
Fence consists of primary shapes: lines, strip
----------
##########
----------
enter 1 for house, 2 for tree, 3 for river, 4 for fence, 5 for hill
5
Hills consists of primary shapes: triangles

   *
  * *
 *   *
*******
   *
  * *
 *   *
*******
   *
  * *
 *   *
*******
Process returned 0 (0x0)   execution time : 6.592 s
Press any key to continue.
```

Coding Details:

We consider each graphical structure as node of a tree, where simple/basic objects are leaf nodes, composite objects are internal nodes. And if we can make an object comprised of all shapes, it can be considered as root node.

1. The Graphic interface describes all methods that both simple and complex shapes can use and improvise.

```
class Graphic {
public:
   virtual void draw() const = 0;
   virtual void remove(Graphic *g) {}
   virtual void add(Graphic *g) {}
   virtual void getChild(int) {}
   virtual ~Graphic() {}
};
```

2. All basic shapes (leaves of tree) are subclasses of the interface that can also be called realizations.

```
class Line : public Graphic {
public:
   void draw() const {
      cout << "----------\n";
   }
};

class Strip : public Graphic {
public:
   void draw() const {
      cout << "##########\n";
   }
};
```

```
class Zigzag : public Graphic {
public:
  void draw() const {
    cout << "_-_-_-_-_-_\n";
  }
};
class Lineup : public Graphic {
public:
  void draw() const {
    cout << " |   |\n |   |\n |   |\n |   |\n |   |\n";
  }
};
class Rectangle : public Graphic {
public:
  void draw() const {
    cout << "\n-------\n|     |\n|     |\n|     |\n|     |\n";
  }
};
class Circle : public Graphic {
public:
  void draw() const {
    cout << "  ***\n *   *\n*     *\n *   *\n  ***\n";
  }
};
```

```
class Triangle : public Graphic {
public:
  void draw() const {
    cout << "\n   *\n  * *\n *   *\n*******";
  }
};
```

3. Picture here is the main composite class which is a subclass of graphic interface and contains the vector named gList. Storing data in the vector can mimic functionality of the trees that we want here.

```
class Picture : public Graphic {
public:
  void draw() const {
    // for each element in gList, call the draw member function
    for_each(gList.begin(), gList.end(), mem_fun(&Graphic::draw));
  }

  void add(Graphic *aGraphic) {
    gList.push_back(aGraphic);
  }
```

Every time we want a composite shape, it adds the basic shapes in the vector and the prints/ draw them one by one to output the desired shapes. So, we don't have to save hundreds different shapes, rather few basic shapes do the desired job.

4. Finally, in the main class or driver class, we instantiate each shape into objects.

```
int main()
{
   cout<< "Primary Objects:\n";
   cout<< "line:\n";
   Line line;
   line.draw();
   cout<< "Line-Upwards:\n";
   Lineup lineup;
   lineup.draw();
   cout<< "Rectangle:\n";
   Rectangle rect;
   rect.draw();
   Rectangle rect;
   rect.draw();
   cout<< "Circle:\n";
   Circle circle;
   circle.draw();
   cout<< "Triangle:\n";
   Triangle triangle;
   triangle.draw();
   cout<< "Zigzag:\n";
   Zigzag zigzag;
   zigzag.draw();
   cout<< "Strips:\n";
   Strip strip;
   strip.draw();
```

Then we take command for various composite shapes. According to the demand, primary shapes are added in the vector in the Picture class (composite class) and then the composite shape is drawn.

```
int n=5;

while(n--){
  cout<<"enter 1 for house, 2 for tree, 3 for river, 4 for fence, 5 for hill\n"
  int p;
  cin>>p;
  if(p==1){
      cout<<"House consists of primary shapes: triangle, rectangle, line\n";
      Picture house;
      house.add(&triangle);
      house.add(&rect);
      house.add(&line);
      house.draw();
  }
  if(p==2){
      cout<<"Tree consists of primary shapes: circle, lineup, line\n";
      Picture tree;
      tree.add(&circle);
      tree.add(&lineup);
      tree.add(&line);
      tree.draw();

  }
  if(p==3){
      cout<<"Water consists of primary shapes: lines, zigzags\n";
      Picture water;
      water.add(&line);
      water.add(&zigzag);
      water.add(&zigzag);
```

```
    if(p==4){
        cout<<"Fence consists of primary shapes: lines, strip\n";
        Picture fence;
        fence.add(&line);
        fence.add(&strip);
        fence.add(&line);
        fence.draw();
    }
    if(p==5){
        cout<<"Hills consists of primary shapes: triangles\n";
        Picture hill;
        hill.add(&triangle);
        hill.add(&triangle);
        hill.add(&triangle);
        hill.draw();
    }
  }
  return 0;
}
```

And thus, the graphical aspect of game is implemented through composite pattern.