1. **模块设计的分析与评价**
2. **附件13.zip是“设计显示一个UI界面中包含的几何图形元素”的C++例子，其中提供了两种设计，主调测试程序为TestUI.cpp。请分别分析评价这两种设计。如果需求改变，如给界面中增加圆形元素，请分别按原设计方法增加这个功能。对比所需修改的部分，分析哪种设计修改量少、通用性高、更适用于软件复用？**
3. 设计composite：在该设计方法下，每一种图形都作为一个独立的类存在。UI类需要为每种图形设置一个单独的数组存储，并且在输出时需要单独输出每一个类。

如果需求改变，给界面中增加其他元素，需要在添加Circle类的基础上，在UI.h中添加Circle的头文件，修改UI类中的成员变量和show函数。

下面是该设计方法下的实现：

Circle.h

1. #ifndef \_CIRCLE\_H
2. #define \_CIRCLE\_H
3. #include "Point.h"
4. #include <iostream>
5. using namespace std;
6. class Circle
7. {
8. public:
9. Circle(const Point &center, float r)
10. {
11. this->center = center;
12. this->r = r;
13. }
14. void show()
15. {
16. cout << "Circle center:(" << center.getX() << "," << center.getY() << ") radius:" << r << endl;
17. }
18. private:
19. Point center;
20. float r;
21. };
22. #endif

UI.h

1. #ifndef \_UI\_H
2. #define \_UI\_H
3. #include <vector>
4. #include "Point.h"
5. #include "Line.h"
6. #include "Rectangle.h"
7. #include "Circle.h"
8. class UI
9. {
10. public:
11. vector<Point> pointVector;
12. vector<Line> lineVector;
13. vector<Rectangle> rectVector;
14. vector<Circle> circleVector;
15. void show()
16. {
17. for (int i = 0; i < pointVector.size(); i++)
18. pointVector[i].show();
19. for (int i = 0; i < lineVector.size(); i++)
20. lineVector[i].show();
21. for (int i = 0; i < rectVector.size(); i++)
22. rectVector[i].show();
23. for (int i = 0; i < circleVector.size(); i++)
24. circleVector[i].show();
25. }
26. };
27. #endif *//\_UI\_H*

TestUi.cpp

1. #include "Point.h"
2. #include "Line.h"
3. #include "Rectangle.h" \*/
4. #include "UI.h"
5. int main(){
6. UI ui;
8. Point p1(1,2),p2(7,8);
9. ui.pointVector.push\_back(p1);
10. ui.pointVector.push\_back(p2);
12. Line line1(p1,p2);
13. ui.lineVector.push\_back(line1);
15. Rectangle rect1(p1,5,6);
16. ui.rectVector.push\_back(rect1);
18. Circle circ1(p1, 3);
19. ui.circleVector.push\_back(circ1);
20. ui.show();
22. return 0;
23. }
24. 设计inherit abstract：在该设计方法下，所有图形都是shape类的子类。UI类只需一个shape类的数组就能存储所有的种类的图形，并且在输出时只需针对shape类进行输出。

如果需求改变，给界面中增加其他元素，则只需添加Circle类，UI类无需改动。

下面是该设计方法下的实现：

Circle.h

1. #ifndef \_CIRCLE\_H
2. #define \_CIRCLE\_H
3. #include "Point.h"
4. #include "Shape.h"
5. #include <iostream>
6. using namespace std;
7. class Circle : public Shape
8. {
9. public:
10. Circle(const Point &center, float r)
11. {
12. this->center = center;
13. this->r = r;
14. }
15. virtual void show()
16. {
17. cout << "Circle center:(" << center.getX() << "," << center.getY() << ") radius:" << r << endl;
18. }
19. private:
20. Point center;
21. float r;
22. };
23. #endif

TestUi.cpp

1. #include "Point.h"
2. #include "Line.h"
3. #include "Rectangle.h"
4. #include "Circle.h"
5. #include "UI.h"
6. main(){
7. UI ui;
9. Point p1(1,2),p2(7,8);
10. ui.shapeVector.push\_back(&p1);
11. ui.shapeVector.push\_back(&p2);
13. ui.shapeVector.push\_back(new Line(p1,p2));
15. ui.shapeVector.push\_back(new Rectangle(p1,5,6));
17. ui.shapeVector.push\_back(new Circle(p1, 3));
18. ui.show();
19. }

对比两种方式，可以发现第二种设计的修改量更少，通用性更高，更适用于软件复用。

1. **实现教材图6-13中的 Engineer与Software Engineer的两种设计实例，并进行讨论。如果要利用组合来实现可替换性，需要怎样设计？**

下面是两种设计的实现，其中SoftwareEngineer1为继承实现，SoftwareEngineer2为组合实现。

1. class Engineer
2. {
3. public:
4. void engCapabilities();
5. };
6. class SoftwareEngineer1 : public Engineer
7. {
8. };
9. class SoftwareEngineer2
10. {
11. public:
12. Engineer engineer;
13. };
14. int main()
15. {
16. SoftwareEngineer1 se1;
17. SoftwareEngineer2 se2;
18. se1.engCapabilities();
19. se2.engineer.engCapabilities();
20. }
21. **参照课本P224图6-14，设计体现德米特法则的实例。分析该种设计的特点?**
22. #include <iostream>
23. #include <string>
24. #include <vector>
26. using namespace std;
28. class Book {
29. private:
30. string title;
31. string author;
32. public:
33. Book(const string& title, const string& author)
34. : title(title), author(author) {}
36. string getTitle() const {
37. return title;
38. }
39. string getAuthor() const {
40. return author;
41. }
42. };
43. class Library {
44. private:
45. vector<Book> books;
46. public:
47. void addBook(const Book& book) {
48. books.push\_back(book);
49. }
50. void displayBooks() const {
51. for (const auto& book : books) {
52. cout << "Title: " << book.getTitle() << ", Author: " << book.getAuthor() << endl;
53. }
54. }
55. };
56. class User {
57. private:
58. string name;
59. public:
60. User(const string& name)
61. : name(name) {}
62. void borrowBook(Library& library, const string& title) {
63. for (auto& book : library.books) {
64. if (book.getTitle() == title) {
65. cout << "User " << name << " borrowed the book: " << book.getTitle() << endl;
66. return;
67. }
68. }
69. cout << "Book with title " << title << " not found." << endl;
70. }
71. };

Library 类负责管理图书，User 类表示用户，而 Book 类表示书籍。User 类只与 Library 类进行交互，通过 Library 类提供的接口来借阅书籍。这样，User 类不需要了解 Book 类的内部实现细节，也不需要直接与 Book 类进行交互。