Object Oriented Programming

Composition

Mr. Usman Wajid

usman.wajid@nu.edu.pk

National University of Computer & Emerging Sciences

Inheritance Vs Composition

The two common ways to relate classes in a meaningful way are:

Inheritance Vs Composition

The two common ways to relate classes in a meaningful way are:

1. **Inheritance:** ("is a" relationship)

Inheritance Vs Composition

The two common ways to relate classes in a meaningful way are:

1. **Inheritance:** (”is a” relationship)
2. **Composition:** (”has-a” relationship)

Composition

Composition

When one or more member(s) of a class are objects of another class type

Composition

Composition

When one or more member(s) of a class are objects of another class type

- In composition two classes ”has a” relationship between them

Composition

When one or more member(s) of a class are objects of another class type

- In composition two classes ”has a” relationship between them
- Similar, to rest of all concepts of OOP, composition prevents us from re-inventing the wheel

Composition

Composition

When one or more member(s) of a class are objects of another class type

- In composition two classes ”has a” relationship between them
- Similar, to rest of all concepts of OOP, composition prevents us from re-inventing the wheel
- Hence, it optimizes code reuse-ability

Composition Example 1

```cpp
class Element {
        int val;
        public:
        int getVal(){ return val; }
        void setVal(int val){ this->val = val;}
};
class Collection {
        Element e1, e2;
        public:
                void setElement(int eNum, int val){
                        if (eNum==1)
                                e1.setVal(val);
                        else
                                e2.setVal(val);
                }
                int getElement(int eNum){
                        if (eNum==1)
                                return e1.getVal();
                        else
                                return e2.getVal();
                }
};
```

```cpp
int main() {
        Collection c;
        for (int i=1; i<=2; i++)
        c.setElement(i, i+1);
        for (int i=1; i<=2; i++)
        {
        cout<<"Element # "<<i;
        cout<<" = ";
        cout<<c.getElement(i);
        }
}
```

Composition Example 1

```cpp
class Element {
        int val;
        public:
        int getVal(){ return val; }
        void setVal(int val){ this->val = val;}
};
class Collection {
        Element e1, e2;
        public:
                void setElement(int eNum, int val){
                        if (eNum==1)
                                e1.setVal(val);
                        else
                                e2.setVal(val);
                }
                int getElement(int eNum){
                        if (eNum==1)
                                return e1.getVal();
                        else
                                return e2.getVal();
                }
};
```

```cpp
int main() {
        Collection c;
        for (int i=1; i<=2; i++)
        c.setElement(i, i+1);
        for (int i=1; i<=2; i++)
        {
        cout<<"Element # "<<i;
        cout<<" = ";
        cout<<c.getElement(i);
        }
}
```

```
Element # 1 = 2
Element # 2 = 3
```

Composition Example 2

```
class Element {
        int val;
public:
        Element(){cout<<"Element constructed!\n";}
        int getVal(){return val;}
        void setVal(int val){this->val = val;}
};
class Collection {
        Element e1, e2;
public:
        Collection(){cout<<"collection constructed!";}
        void setElement(int eNum, int val){
        if (eNum==1)
                e1.setVal(val);
        else
                e2.setVal(val);}
        int getElement(int eNum){
                if (eNum==1)
                        return e1.getVal();
                else
                        return e2.getVal();}
};
```

```
int main() {
        Collection c;
}
```

Composition Example 2

```
class Element {
        int val;
public:
        Element(){cout<<"Element constructed!\n";}
        int getVal(){return val;}
        void setVal(int val){this->val = val;}
};
class Collection {
        Element e1, e2;
public:
        Collection(){cout<<"collection constructed!";}
        void setElement(int eNum, int val){
        if (eNum==1)
                e1.setVal(val);
        else
                e2.setVal(val);}
        int getElement(int eNum){
                if (eNum==1)
                        return e1.getVal();
                else
                        return e2.getVal();}
};
```

```
int main() {
        Collection c;
}
```

```
Element constructed!
Element constructed!
collection constructed!
```

Composition Example 3

```
class Element {
        int val;
        public:
        Element(int x)
        {       setVal(x);
                cout<<"Element("<<x<<") constructed!\n";}
        int getVal(){return val;}
        void setVal(int val){this->val = val;}
};
class Collection {
        Element e1, e2;
        public:
        Collection(): e2(2), e1(1) {
                cout<<"collection constructed!";
        }
        void setElement(int eNum, int val){
                if (eNum==1)
                e1.setVal(val);
                else
                e2.setVal(val);}
        int getElement(int eNum){
                if (eNum==1)
                return e1.getVal();
                else
                return e2.getVal();}
};
```

```
int main() {
        Collection c;
}
```

Composition Example 3

```
class Element {
        int val;
        public:
        Element(int x)
        {       setVal(x);
                cout<<"Element("<<x<<") constructed!\n";}
        int getVal(){return val;}
        void setVal(int val){this->val = val;}
};
class Collection {
        Element e1, e2;
        public:
        Collection(): e2(2), e1(1) {
                cout<<"collection constructed!";
        }
        void setElement(int eNum, int val){
                if (eNum==1)
                e1.setVal(val);
                else
                e2.setVal(val);}
        int getElement(int eNum){
                if (eNum==1)
                return e1.getVal();
                else
                return e2.getVal();}
};
```

```
int main() {
        Collection c;
}
```

```
Element(1) constructed!
Element(2) constructed!
collection constructed!
```

Composition continued . . .

when the constructor is divided between the declaration and the definition, the list of alternative constructors should be associated with the definition, not the declaration

- This means that the following code snippet is correct,

```
class X {
        public:
        X(int x) { };
};
class Y {
        X x;
        public:
        Y(int x);
};
Y::Y(int x) : x(1) { };
```

Composition continued ...

Member-objects of a class are constructed:

- In the order/sequence they are declared

Composition continued . . .

Member-objects of a class are constructed:

- In the order/sequence they are declared
- Not in the order they are listed in the constructor's member initialization list

Composition continued . . .

Member-objects of a class are constructed:

- In the order/sequence they are declared
- Not in the order they are listed in the constructor's member initialization list
- Before the enclosing class objects are constructed

Composition Example 4

```
class A{
        private:
                int y;
        public:
                int x;
                A (int a, int b) {x=a; y=b;}
                void print(){
                        cout<<"x = "<<x<<"\ty = "<<y<<endl;
                }
};
class B{
        private:
                int z;
        public:
                A objA;
                B(int a, int b, int c): objA(a,b) {z = c;}
                void print(){
                        objA.print();
                        cout<<"z = "<<z<<endl;
                }
};
```

```
int main() {

        B objB(3, 4, 5);
        objB.print();
        objB.objA.print();

}
```

Composition Example 4

```
class A{
        private:
                int y;
        public:
                int x;
                A (int a, int b) {x=a; y=b;}
                void print(){
                        cout<<"x = "<<x<<"\ty = "<<y<<endl;
                }
};
class B{
        private:
                int z;
        public:
                A objA;
                B(int a, int b, int c): objA(a,b) {z = c;}
                void print(){
                        objA.print();
                        cout<<"z = "<<z<<endl;
                }
};
```

Initialization list goes with the constructor contacting composition in order to pass parameters to the other class constructor

```
int main() {

        B objB(3, 4, 5);
        objB.print();
        objB.objA.print();

}
```

Composition Example 4

```cpp
class A{
        private:
                int y;
        public:
                int x;
                A (int a, int b) {x=a; y=b;}
                void print(){
                        cout<<"x = "<<x<<"\ty = "<<y<<endl;
                }
};
class B{
        private:
                int z;
        public:
                A objA;
                B(int a, int b, int c): objA(a,b) {z = c;}
                void print(){
                        objA.print();
                        cout<<"z = "<<z<<endl;
                }
};
```

Initialization list goes with the constructor contacting composition in order to pass parameters to the other class constructor

```cpp
int main() {

        B objB(3, 4, 5);
        objB.print();
        objB.objA.print();

}
```

```
x = 3    y = 4
z = 5
x = 3    y = 4
```