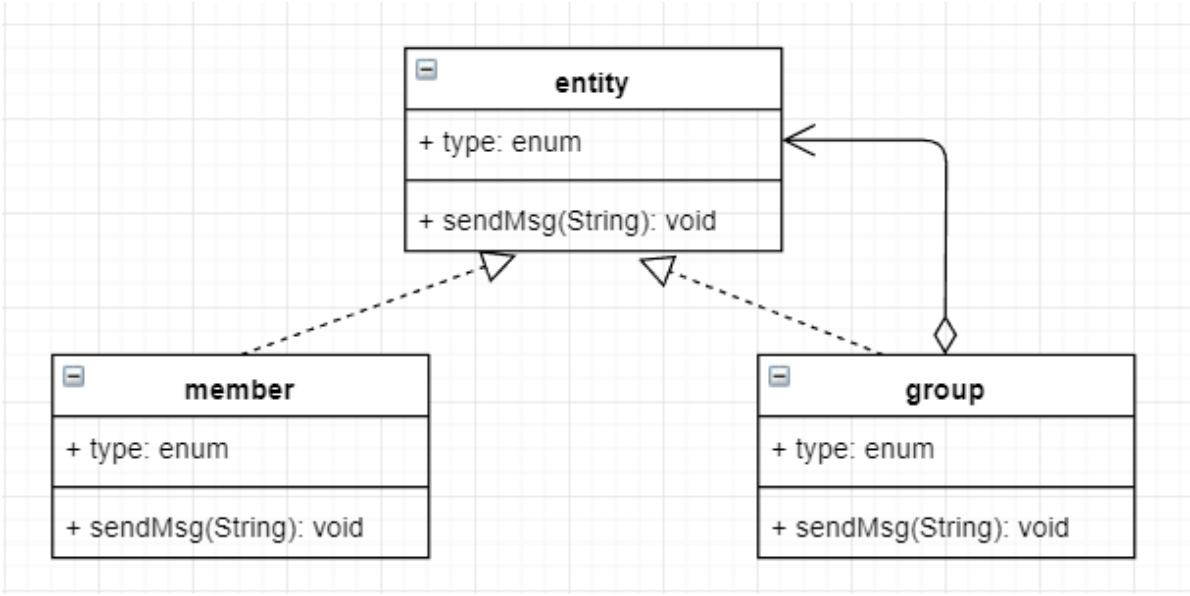
设计模式作业4

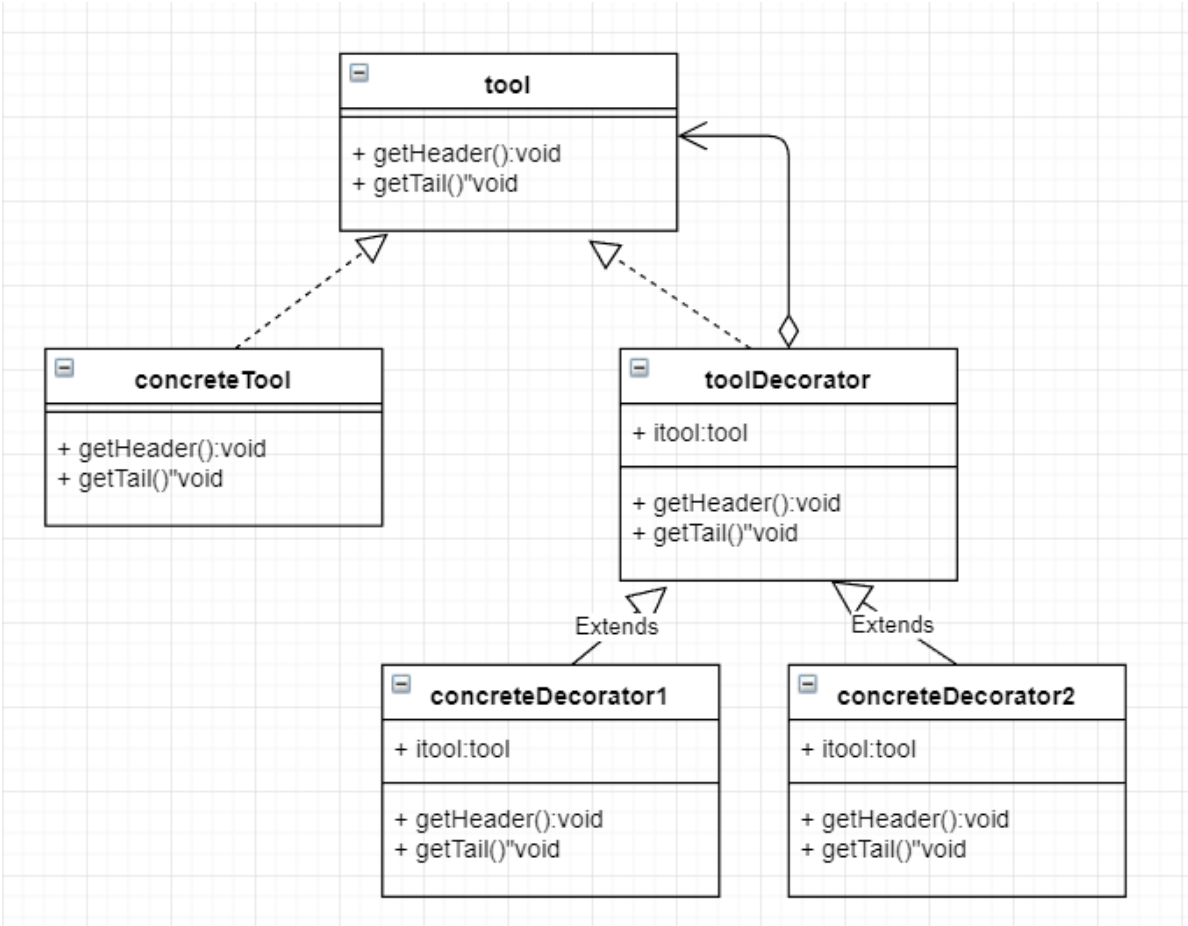
1.


entity
+ type: enum
+ sendMsg(String): void
member
+ type: enum
+ sendMsg(String): void
group
+ type: enum
+ sendMsg(String): void

```java
public class client {
    public static void main(String[] args) {
        group group1 = new group();
        group group2 = new group();
        member member1 = new member();
        member member2 = new member();
        member member3 = new member();

        group1.addMember(member1);
        group2.addMember(member2);
        group2.addMember(member3);
        group1.addMember(group2);
        group1.senMsg("sssssss");

    }
}
public interface entity {
    5 usages
    enum eType{member,group};

    eType type = null;
    2 usages  2 implementations
    public void senMsg(String e);
}
4 usages
public class group implements entity{
    2 usages
    List<entity> groupMember = new ArrayList<>();
    1 usage
    eType type = eType.group;

    4 usages
    public void addMember(entity e) { this.groupMember.add(e); }

    2 usages
    @Override
    public void senMsg(String msg) {
        for (entity e:this.groupMember)
        {
            e.senMsg(msg);
        }
        System.out.println("msg" + this.type);
    }
}
```

```java
public class member implements entity{

    1 usage
    eType type = eType.member;
    2 usages
    @Override
    public void senMsg(String e) {
        System.out.println(e + type);

    }
}
```

2.



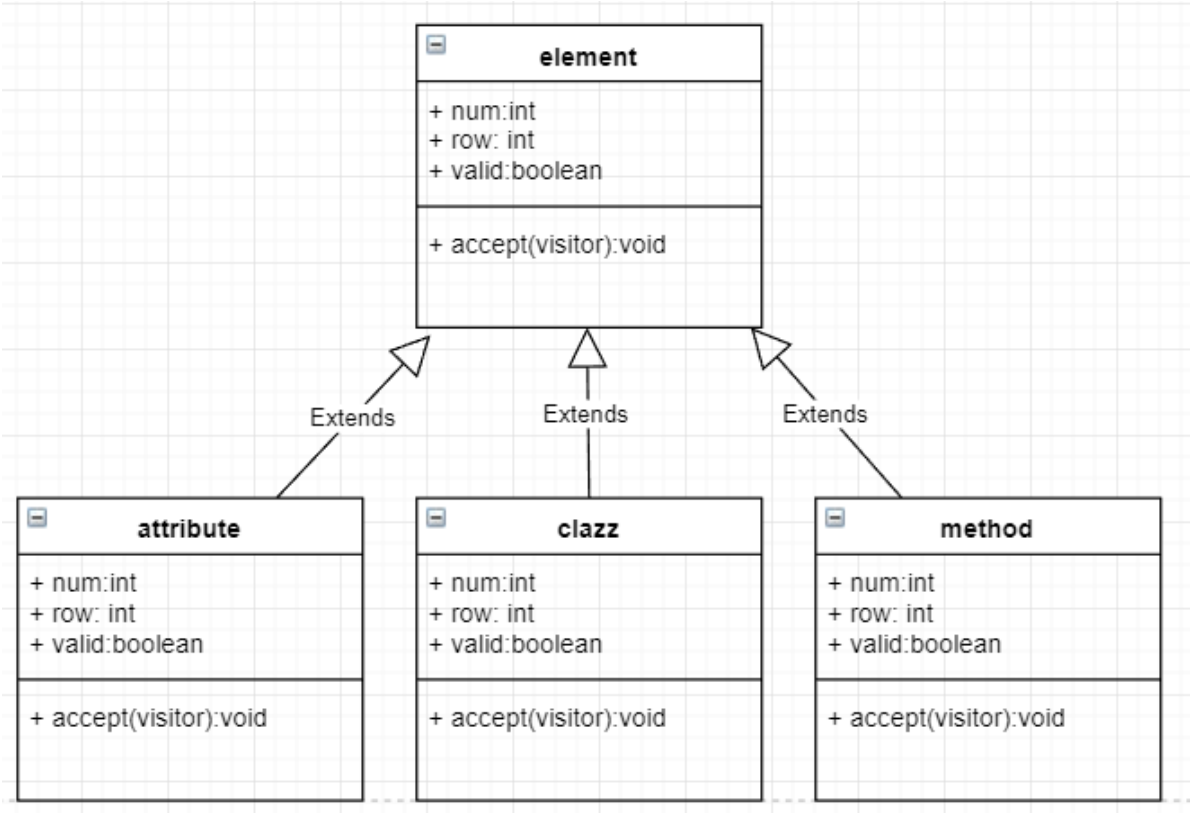
```java
public class client {
    public static void main(String[] args) {
        Itool itool = new tool1();
        itool.setContent("sadadasdasdas");

        itool.genHeader();
        itool.getContent();
        itool.genTail();
        System.out.println(itool.getContent());
        Itool itool1 = new tool2(itool);

        itool1.genHeader();
        itool1.getContent();
        itool1.genTail();
        System.out.println(itool1.getContent());
    }
}
```

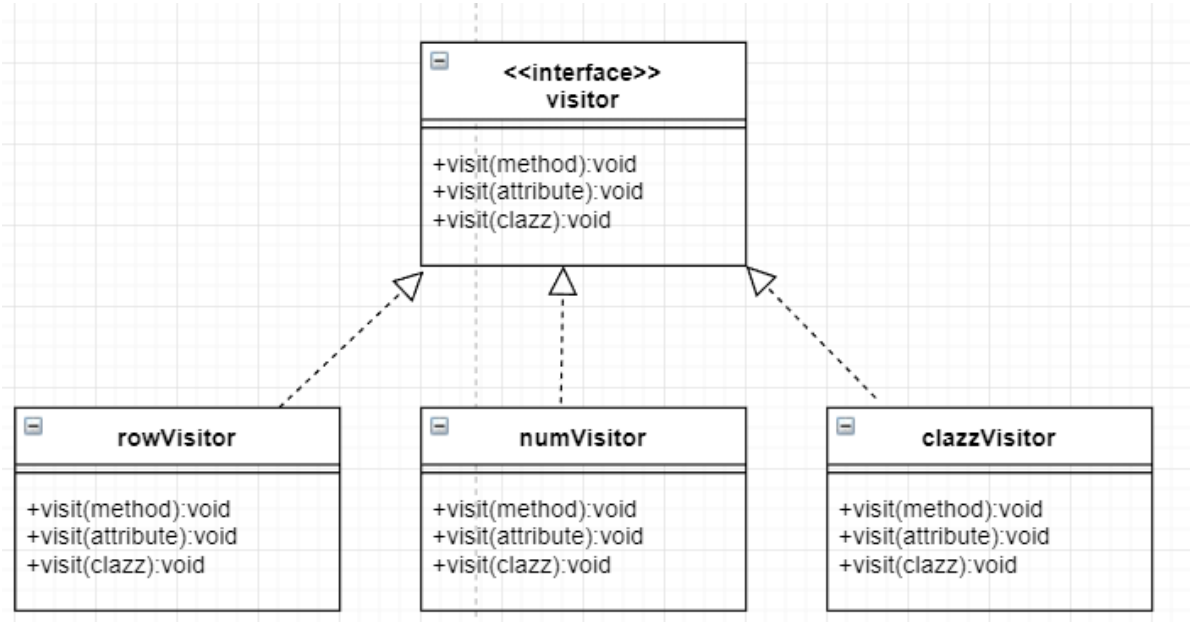
```java
public abstract class Itool {
    1 usage
    Itool itool = null;
    7 usages
    String content = null;
    2 usages
    Itool(){}
    2 usages
    Itool(Itool itool1)
    {
        this.itool = itool1;
        this.content = itool1.getContent();
    }

    1 usage
    public void setContent(String content) { this.content = content; }

    9 usages
    public String getContent() { return content; }

    2 usages  2 implementations
    public abstract void genHeader();

    2 usages  2 implementations
    public abstract void genTail();
}
```

```java
public class tool1 extends Itool{

    tool1(Itool itool1) {
        super(itool1);
    }
    1 usage
    tool1(){}
    2 usages
    @Override
    public void genHeader(){
        this.content="tool1H" + this.getContent();
    }
    2 usages
    @Override
    public void genTail(){
        this.content = this.getContent() +"tool1T";
    }
}
```

```java
public class tool2 extends Itool{
    1 usage
    tool2(Itool itool1) {
        super(itool1);
    }
    tool2(){}
    2 usages
    @Override
    public void genHeader(){
        this.content="tool2H" + this.getContent();
    }
    2 usages
    @Override
    public void genTail(){
        this.content = this.getContent() +"tool2T";
    }

}
```

3.


element
+ num:int
+ row: int
+ valid:boolean
+ accept(visitor):void
Extends
Extends
Extends
attribute
+ num:int
+ row: int
+ valid:boolean
+ accept(visitor):void
clazz
+ num:int
+ row: int
+ valid:boolean
+ accept(visitor):void
method
+ num:int
+ row: int
+ valid:boolean
+ accept(visitor):void


<<interface>>
visitor
+visit(method):void
+visit(attribute):void
+visit(clazz):void
rowVisitor
+visit(method):void
+visit(attribute):void
+visit(clazz):void
numVisitor
+visit(method):void
+visit(attribute):void
+visit(clazz):void
clazzVisitor
+visit(method):void
+visit(attribute):void
+visit(clazz):void

```java
public class client {
    public static void main(String[] args) {
        numVisitor numvisitor = new numVisitor();
        rowVisitor rowvisitor = new rowVisitor();
        validVisitor validvisitor = new validVisitor();
        atrribute atrribute1 = new atrribute();
        clazz c1 = new clazz();
        method m1 = new method();

        atrribute1.accept(numvisitor);
        atrribute1.accept(rowvisitor);
        atrribute1.accept(validvisitor);

        c1.accept(numvisitor);
        c1.accept(rowvisitor);
        c1.accept(validvisitor);

        m1.accept(numvisitor);
        m1.accept(rowvisitor);
        m1.accept(validvisitor);
    }
```

```java
public class atrribute extends element{
    1 usage
    public atrribute() {
        this.num = 100;
        this.row = 1000;
        this.valid = true;
    }

    9 usages
    @Override
    public void accept(visitor v) {
        v.visit( a: this);
    }
}
```

```java
public class clazz extends element{
    1 usage
    public clazz() {
        this.num = 101;
        this.row = 1001;
        this.valid = true;
    }
    9 usages
    @Override
    public void accept(visitor v) {
        v.visit( c: this);
    }
}
```

```java
public abstract class element {

    6 usages
    public int num;
    6 usages
    public boolean valid;
    public int row;

    9 usages  3 implementations
    public abstract void accept(visitor v);
}

public class method extends element{
    1 usage
    public method() {
        this.num = 102;
        this.row = 1002;
        this.valid = true;
    }
    9 usages
    @Override
    public void accept(visitor v) {
        v.visit( m: this);
    }
}
```

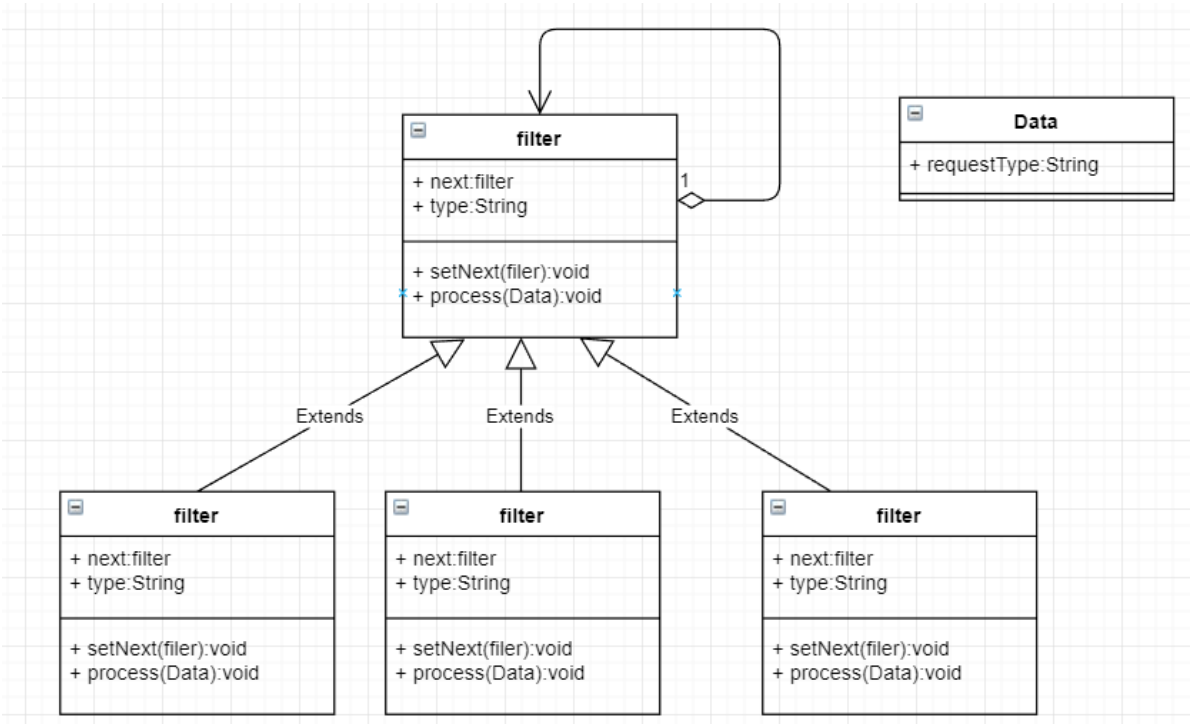
```
public class numVisitor implements visitor {
    1 usage
    @Override
    public void visit(clazz c) {
        System.out.println("clazz num: " + String.valueOf(c.num));

    }

    1 usage
    @Override
    public void visit(method m) {
        System.out.println("method num: " + String.valueOf(m.num));

    }

    1 usage
    @Override
    public void visit(atrribute a) { System.out.println("atrribute num: " + String.valueOf(a.num)); }
}
```

```
public class rowVisitor implements visitor{
    1 usage
    @Override
    public void visit(clazz c) {
        System.out.println("clazz row: " + String.valueOf(c.row));

    }

    1 usage
    @Override
    public void visit(method m) {
        System.out.println("method row: " + String.valueOf(m.row));

    }

    1 usage
    @Override
    public void visit(atrribute a) {
        System.out.println("atrribute row: " + String.valueOf(a.row));
    }
}
```

```
public class validVisitor implements visitor{
    1 usage
    @Override
    public void visit(atrribute a) {
        System.out.println("attr valid:" + String.valueOf(a.valid));
    }

    1 usage
    @Override
    public void visit(clazz c) {
        System.out.println("clazz valid:" + String.valueOf(c.valid));

    }
```

```
    }

    1 usage
    @Override
    public void visit(method m) {
        System.out.println("method valid:" + String.valueOf(m.valid));
    }
}
public interface visitor {
    1 usage  3 implementations
    public void visit(atrribute a);

    1 usage  3 implementations
    public void visit(clazz c);

    1 usage  3 implementations
    public void  visit(method m);

}
```

4.



```
public class charSetFilter extends filter{
    1 usage
    public charSetFilter() {
        this.filerType = "charSet";
    }

    @Override
    public void process(Data data) {
        if(data.requestType.equals(this.filerTyp
        {
            System.out.println("char set");
        }
        else
        {
            this.next.process(data);
        }
    }
}
public class client {
    public static void main(String[] args) {
        charSetFilter charSetFilter = new charSetFilter();
        charSetFilter.next = new dataCheckFilter();
        charSetFilter.next.next = new dataTypeFilter();


        Data data = new Data();
        data.setRequestType("dataCheck");
        charSetFilter.process(data);


        data.setRequestType("charSet");
        charSetFilter.process(data);


        data.setRequestType("dataType");
        charSetFilter.process(data);
    }
}
public class Data {
    5 usages
    public String requestType;

    public String getRequestType() {
        return requestType;
    }

    3 usages
```

```
    public void setRequestType(String requestType) {
        this.requestType = requestType;
    }

}
public class dataCheckFilter extends filter{
    1 usage
    public dataCheckFilter() {
        this.filerType = "dataCheck";
    }

    @Override
    public void process(Data data) {
        if(data.requestType.equals(this.filerType))
        {
            System.out.println("data check");
        }
        else
        {
            this.next.process(data);
        }
    }
}
```

```java
public class dataTypeFilter extends filter{
    1 usage
    public dataTypeFilter() {
        this.filerType = "dataType";
    }

    @Override
    public void process(Data data) {
        if(data.requestType.equals(this.filerType)
        {
            System.out.println("data type");
        }
        else
        {
            this.next.process(data);
        }
    }
}
public abstract class filter {

    protected filter next;

    6 usages
    String filerType;

    public filter getNext() { return next; }

    public void setNext(filter next) { this.next = next; }

    3 implementations
    public abstract void process(Data data);
}
```

实验小结

使用四种设计模式对不同的情况进行建模编码过程，总结如下：

1.composite模式：

将容器和内容作为同一种东西看待，可以方便我们递归的处理问题，在容器中既可以放入容器，又可以放入内容，然后在小容器中，又可以继续放入容器和内容，这样就构成了容器结构和递归结构。将对象组合成树形结构以表示 “部分-整体” 的层次结构。Composite 使得用户对于单个对象和组合对象的使用具有一致性。

2.装饰器模式：

允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

1. 在不影响其它对象的情况下，以动态、透明的方式给单个对象添加职责
2. 处理那些可以撤销的职责

3. 当使用子类扩展的方式不切实际的时候，可考虑使用装饰器模式
4. Java IO包中的类设计运用了装饰器模式

3.访问者模式：

访问者模式是一种将数据操作和数据结构分离的设计模式。

1. 对象结构比较稳定，但经常需要在此对象结构上定义新的操作。
2. 需要对一个对象结构中的对象进行很多不同的并且不相关的操作，而需要避免这些操作“污染”这些对象的类，也不希望在增加新操作时修改这些类。

4.责任链模式：

为了避免请求发送者与多个请求处理者耦合在一起，将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止。