**Visitor** – поведінковий  
Описує операцію, яка виконується з кожним об'єктом з деякої структури. Патерн відвідувач дозволяє визначити нову операцію, не змінюючи класи цих об'єктів.

Використовується коли необхідно для декількох класів зробити подібну (одну і ту ж) операцію.

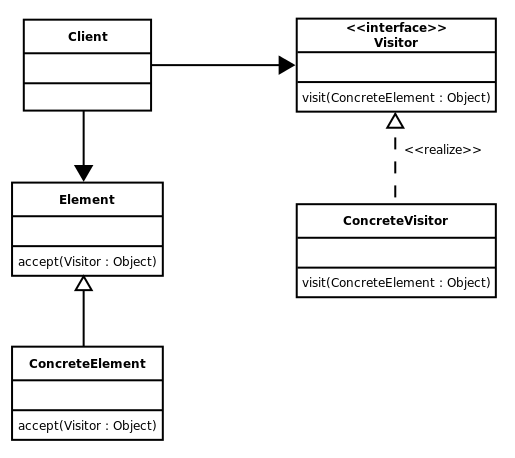


Рис.1 – Структура

1. Додати метод accept (Visitor) в ієрархію «елемент».

2. Створити базовий клас Visitor і визначити методи visit () для кожного типу елемента.

3. Створити похідні класи Visitor для кожної операції, що виконується над елементами.

4. Клієнт створює об'єкт Visitor і передає його в метод, що викликається accept ().

**Переваги:**

спрощується додавання нових операцій;

об'єднання споріднених операції в класі Visitor;

клас Visitor може запам’ятати в якийсь стан по ходу обходу контейнера.

**Недоліки:**

Ускладнене додавання нових класів, оскільки потрібно оновлювати ієрархію відвідувача і його синів.

**Реалізація**

**public** **class** **Demo** {

**public** **static** void main ( String [] args ) {

Point p = **new** Point2d( 1, 2 );

Visitor v = **new** Chebyshev();

p.accept( v );

System.out.println( p.getMetric() );

}

}

**interface** **Visitor** {

**public** void visit ( Point2d p );

**public** void visit ( Point3d p );

}

**abstract** **class** **Point** {

**public** **abstract** void accept ( Visitor v );

**private** double metric = -1;

**public** double getMetric () {

**return** metric;

}

**public** void setMetric ( double metric ) {

**this**.metric = metric;

}

}

**class** **Point2d** **extends** Point {

**public** Point2d ( double x, double y ) {

**this**.x = x;

**this**.y = y;

}

**public** void accept ( Visitor v ) {

v.visit( **this** );

}

**private** double x;

**public** double getX () { **return** x; }

**private** double y;

**public** double getY () { **return** y; }

}

**class** **Point3d** **extends** Point {

**public** Point3d ( double x, double y, double z ) {

**this**.x = x;

**this**.y = y;

**this**.z = z;

}

**public** void accept ( Visitor v ) {

v.visit( **this** );

}

**private** double x;

**public** double getX () { **return** x; }

**private** double y;

**public** double getY () { **return** y; }

**private** double z;

**public** double getZ () { **return** z; }

}

**class** **Euclid** **implements** Visitor {

**public** void visit ( Point2d p ) {

p.setMetric( Math.sqrt( p.getX()\*p.getX() + p.getY()\*p.getY() ) );

}

**public** void visit ( Point3d p ) {

p.setMetric( Math.sqrt( p.getX()\*p.getX() + p.getY()\*p.getY() + p.getZ()\*p.getZ() ) );

}

}

**class** **Chebyshev** **implements** Visitor {

**public** void visit ( Point2d p ) {

double ax = Math.abs( p.getX() );

double ay = Math.abs( p.getY() );

p.setMetric( ax>ay ? ax : ay );

}

**public** void visit ( Point3d p ) {

double ax = Math.abs( p.getX() );

double ay = Math.abs( p.getY() );

double az = Math.abs( p.getZ() );

double max = ax>ay ? ax : ay;

**if** ( max<az ) max = az;

p.setMetric( max );

}

}

**Singleton** – породжуючий

Гарантує, що один і той самий об’єкт буде створений тільки 1 раз і дає глобальну точку доступу до цього об’єкта (в основному класа)

**Використовується тоді, коли:**

* Використовується існуюча об’єктно – орієнтована бібліотека;
* Існує шанс, що 1 об’єкт коли – небудь перетвориться в декілька;
* Інтерфейс об’єкта занадто складний і не варто засмічувати основний пекедж великою кількістю функцій (в основному в іграх);
* В залежності від яких – небудь умов і налаштувань, створюється 1 з декількох об’єктів. Напр. в залежності від того, чи ведеться лог чи ні, створюється або справжній об’єкт, який записує у файл, або заглушка яка нічого не робить;

Такі об’єкти можна створити при ініціалізації програми. Це може призвести до таких проблем:

* Якщо об’єкт потрібен уже при ініціалізаціх, він може бути потрібен і раніше, ніж він буде створений
* Бувають випадки, що об’єкт потрібен не завжди. В такому випадку його створення можна пропустити.

**Плюс**Контролюється доступ до єдиного екземпляру

**Мінуси**Глобальні об’єкти будуть видимі всюди, що ускладнює написання великого проекту  
Ускладнює написання тестів.

**Реалізація**

**public** **class** **Singleton** {

**private** **static** Singleton instance;

**private** Singleton () {}

**public** **static** Singleton getInstance() {

**if** (instance == **null**) {

instance = **new** Singleton();

}

**return** instance;

}

}