在访问者模式（Visitor Pattern）中，我们使用了一个访问者类，它改变了元素类的执行算法。通过这种方式，元素的执行算法可以随着访问者改变而改变。这种类型的设计模式属于行为型模式。根据模式，元素对象已接受访问者对象，这样访问者对象就可以处理元素对象上的操作。

**意图：**主要将数据结构与数据操作分离。

**主要解决：**稳定的数据结构和易变的操作耦合问题。

**何时使用：**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，而需要避免让这些操作"污染"这些对象的类，使用访问者模式将这些封装到类中。

**如何解决：**在被访问的类里面加一个对外提供接待访问者的接口。

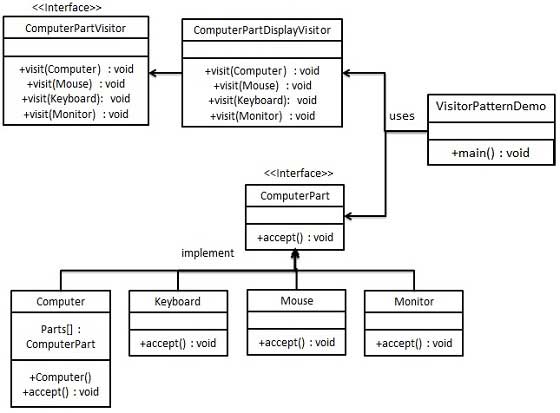
**关键代码：**在数据基础类里面有一个方法接受访问者，将自身引用传入访问者。

**应用实例：**您在朋友家做客，您是访问者，朋友接受您的访问，您通过朋友的描述，然后对朋友的描述做出一个判断，这就是访问者模式。

**缺点：** 1、具体元素对访问者公布细节，违反了迪米特原则。 2、具体元素变更比较困难。 3、违反了依赖倒置原则，依赖了具体类，没有依赖抽象。

我们将创建一个定义接受操作的 *ComputerPart* 接口。*Keyboard*、*Mouse*、*Monitor* 和 *Computer* 是实现了 *ComputerPart* 接口的实体类。我们将定义另一个接口 *ComputerPartVisitor*，它定义了访问者类的操作。*Computer* 使用实体访问者来执行相应的动作。

*VisitorPatternDemo*，我们的演示类使用 *Computer*、*ComputerPartVisitor* 类来演示访问者模式的用法。



ComputerPart computer = new Computer();

computer.accept(new ComputerPartDisplayVisitor());

// 被访问者

public class Computer implements ComputerPart {

ComputerPart[] parts;

public Computer(){

parts = new ComputerPart[] {new Mouse(), new Keyboard(), new Monitor()};

}

@Override

public void accept(ComputerPartVisitor computerPartVisitor) {

for (int i = 0; i < parts.length; i++) {

parts[i].accept(computerPartVisitor);

}

// 访问者为Computer提供的一个访问Computer的接口，如果Computer接受访问者，则调用这个接口，让访问者访问它

computerPartVisitor.visit(this);

}

}