Paul Kummer

Dr. Hanku Lee

Compilers CSIS 455-01

09/10/2021

Lab 01: Examining Visitors

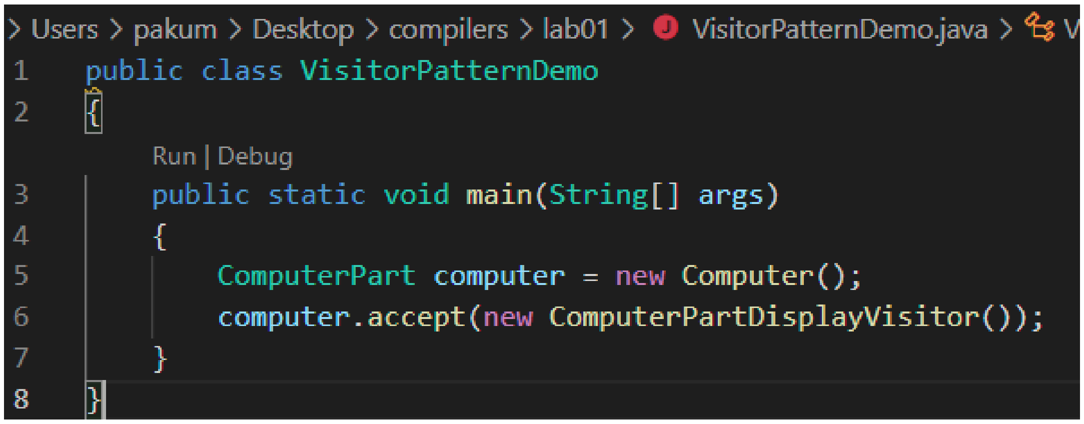
Purpose of Lab

The purpose of the lab was to introduce the design concept of visitors. With visitors, operations can be easily added, and complexity is reduced by containing everything in the visitor instead of many classes. New objects can be added more easily because less code will have to be changed. Visitors also help keep related operations grouped together and hide information within the visitors.

Example Code

VisitorPatternDemo.java

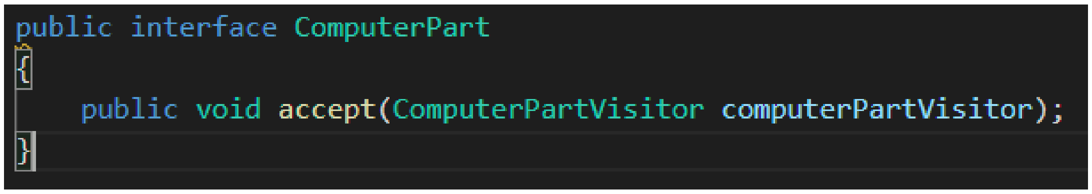
Runs the java code that uses visitors to make a computer with parts. The details of the new computer are hidden in the visitor and are accessed with “computer.accept”.



```
> Users > pakum > Desktop > compilers > lab01 > VisitorPatternDemo.java > V
public class VisitorPatternDemo
{
    Run | Debug
    public static void main(String[] args)
    {
        ComputerPart computer = new Computer();
        computer.accept(new ComputerPartDisplayVisitor());
    }
}
```

ComputerPart.java

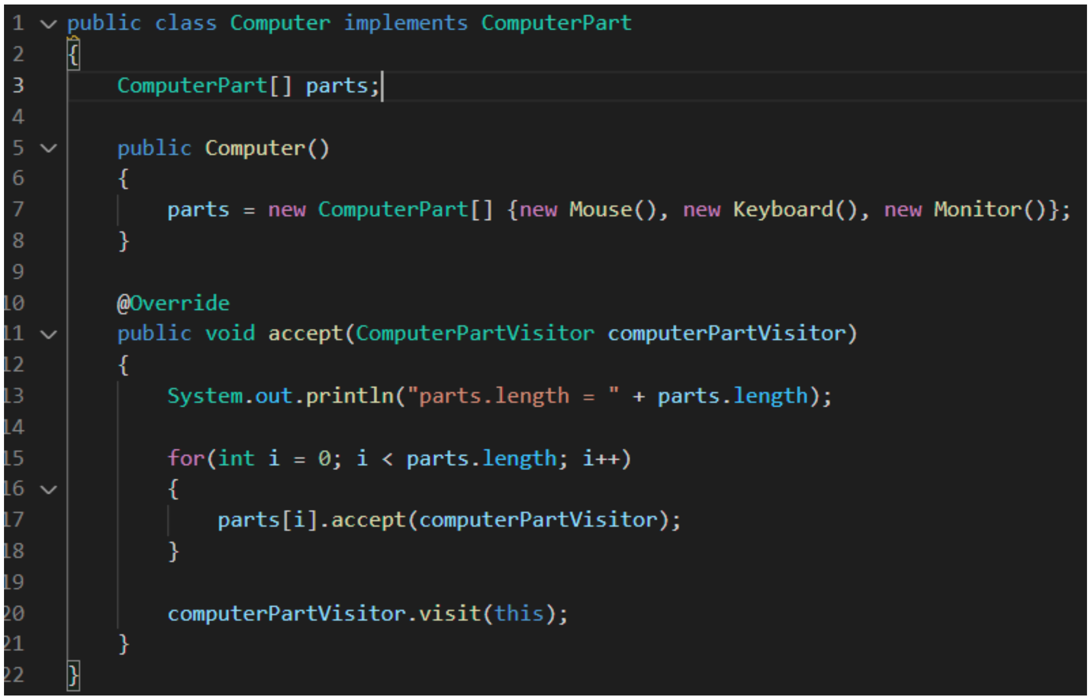
The interface that allows a computer part to accept a visitor.



```
public interface ComputerPart
{
    public void accept(ComputerPartVisitor computerPartVisitor);
}
```

Computer.java

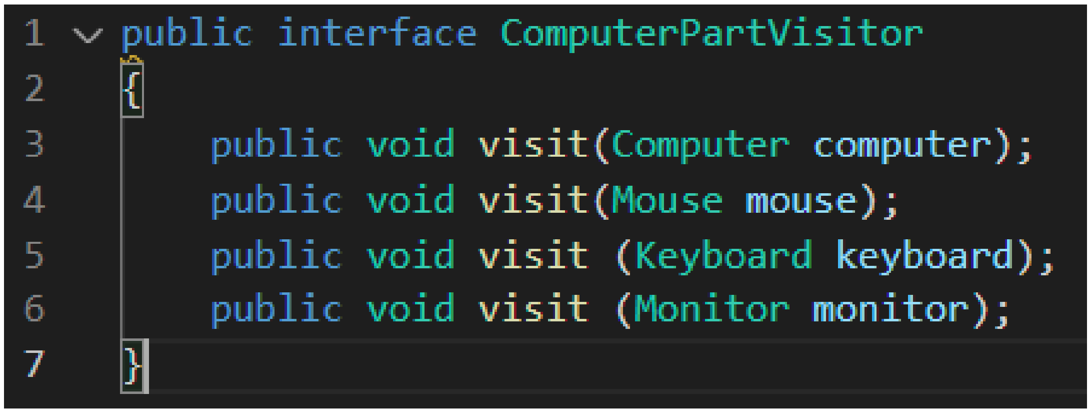
The computer is made up of computer parts, so it inherits the interface of Computer part, allowing use of the visitor. Then each visitors can be accessed with eh accept method.



```java
public class Computer implements ComputerPart
{
    ComputerPart[] parts;

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

    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        System.out.println("parts.length = " + parts.length);

        for(int i = 0; i < parts.length; i++)
        {
            parts[i].accept(computerPartVisitor);
        }

        computerPartVisitor.visit(this);
    }
}
```

ComputerPartVisitor.java

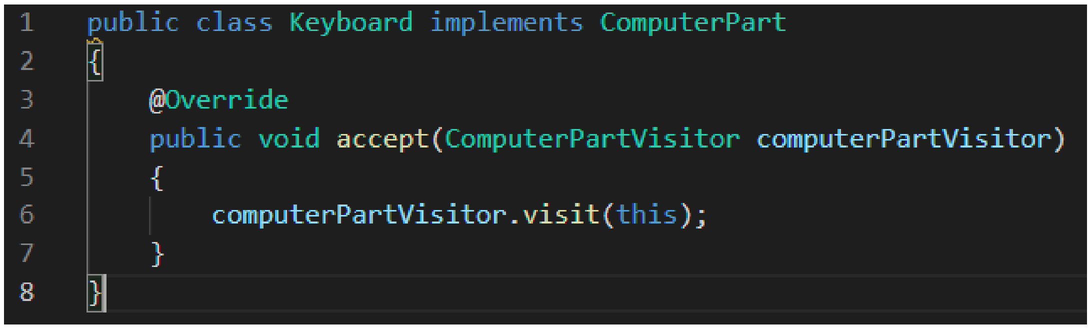
This interface has the overloaded operator of visit, allowing the many different computer parts to be implemented by the visitor.



```java
public interface ComputerPartVisitor
{
    public void visit(Computer computer);
    public void visit(Mouse mouse);
    public void visit (Keyboard keyboard);
    public void visit (Monitor monitor);
}
```

Keyboard.java

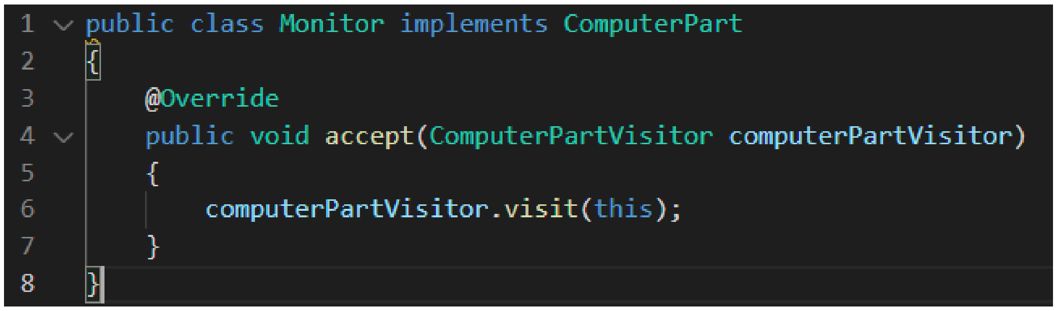
This is a keyboard computer part and overrides the computer part’s default accept method.



```java
public class Keyboard implements ComputerPart
{
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

Monitor.java

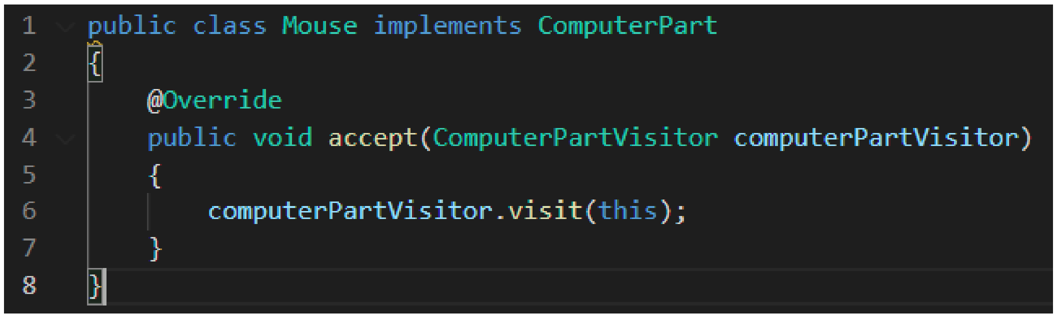
This is a monitor computer part and overrides the computer part’s default accept method.



```java
public class Monitor implements ComputerPart
{
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

Mouse.java

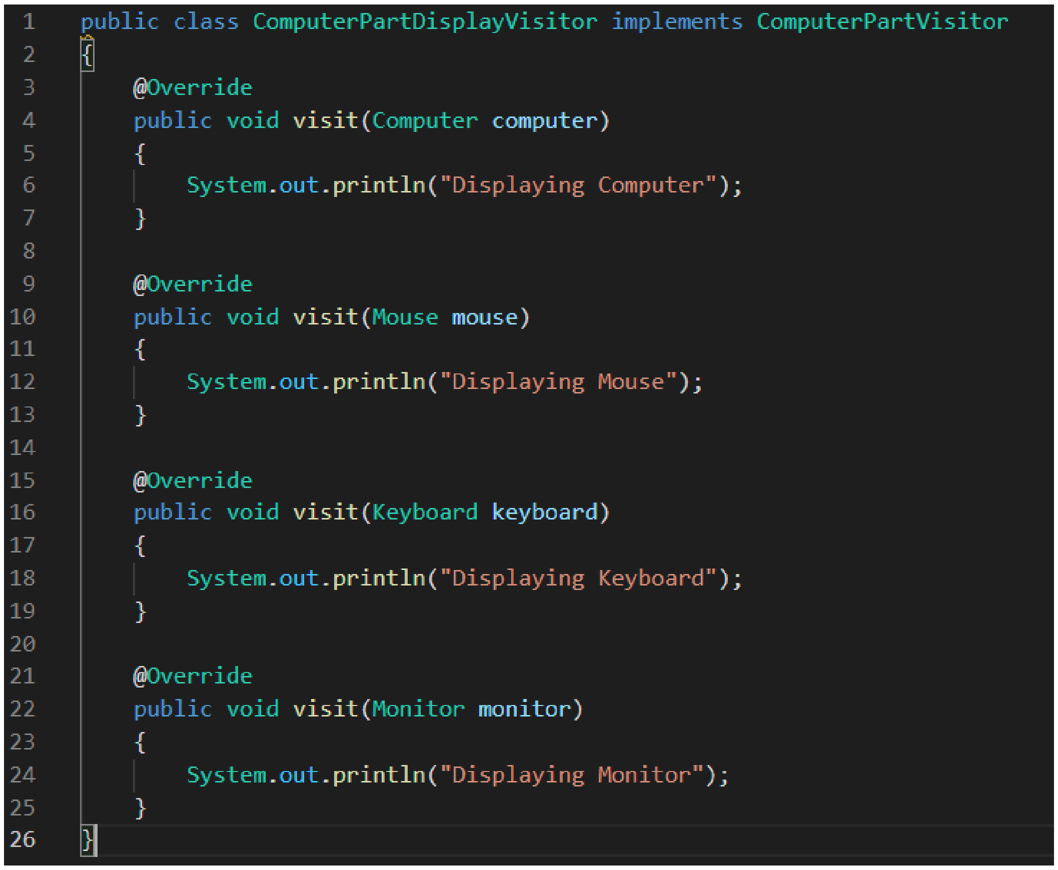
This is a mouse computer part and overrides the computer part’s default accept method.



```java
public class Mouse implements ComputerPart
{
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

ComputerPartDisplayVisitor.java

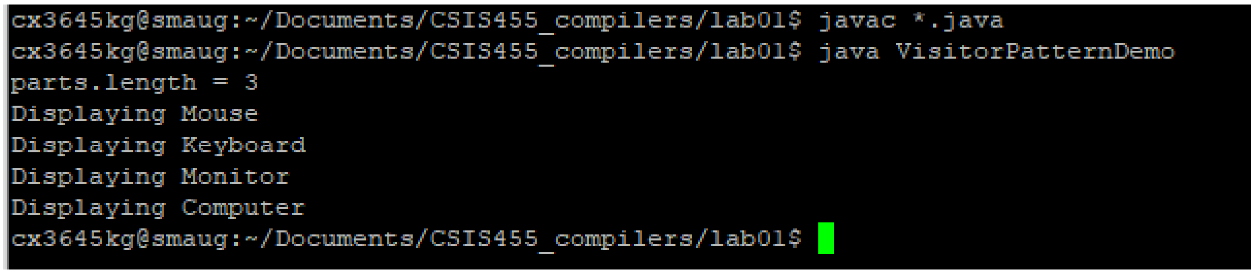
This code will override all the visit methods for ComputerPartVisitor to display what each part is. When the parts are iterated over, the visitor will take on a signature of one of computer parts. This will give each different computer part it's own output.



```java
public class ComputerPartDisplayVisitor implements ComputerPartVisitor
{
    @Override
    public void visit(Computer computer)
    {
        System.out.println("Displaying Computer");
    }

    @Override
    public void visit(Mouse mouse)
    {
        System.out.println("Displaying Mouse");
    }

    @Override
    public void visit(Keyboard keyboard)
    {
        System.out.println("Displaying Keyboard");
    }

    @Override
    public void visit(Monitor monitor)
    {
        System.out.println("Displaying Monitor");
    }
}
```

Code Execution

The computer will have it's accept method called first, which will display how many parts it has. Then each part is iterated over. As they are iterated over, each parts/visitor will have it's accept method called. When accept is called on a visitor that is a computer part, the visit method will be called and the part will output to the terminal with its overloaded visit method.



```
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab01$ javac *.java
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab01$ java VisitorPatternDemo
parts.length = 3
Displaying Mouse
Displaying Keyboard
Displaying Monitor
Displaying Computer
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab01$
```

Conclusion

This lab showed how an abstract syntax tree can be created with the use of the visitor design pattern. Using visitors can allow the simplification of large complex programs that have many different objects with different interfaces. Related operations can be grouped together, preventing classes from containing operations that are unrelated to them.