Paul Kummer

Dr. Hanku Lee

Compilers CSIS 455-01

09/11/2021

Lab 02: Examining Visitors Continued

Purpose of Lab

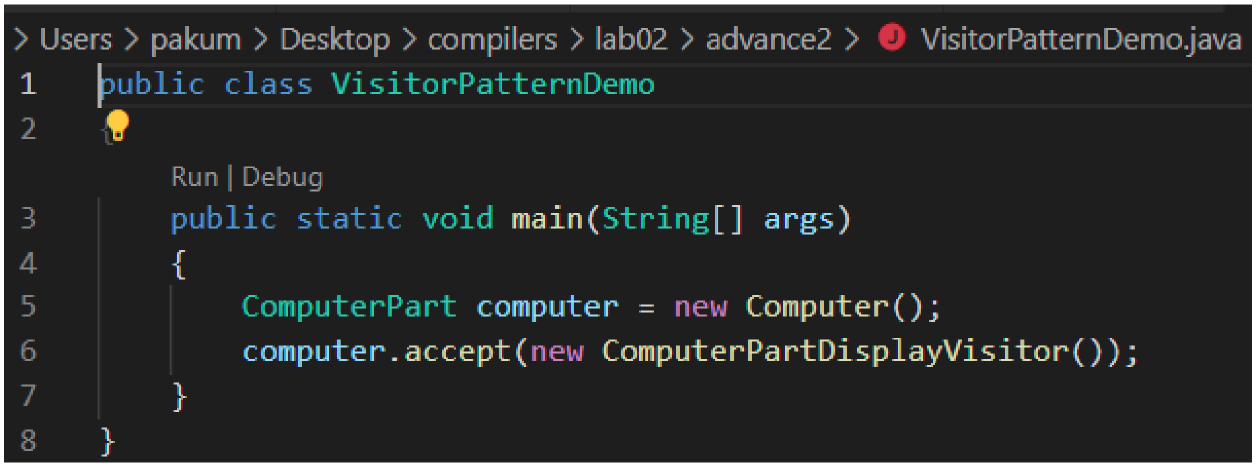
The purpose of this lab is to help clarify the concepts of the previous lab. It shows how a visitor is created and can have information about the visitor retrieved. More importantly, this lab is suppose to demonstrate how using visitors an abstract syntax tree can be created and traversed by calling visit and accept methods and also using interfaces for visitors to interact with other objects.

Example Code

Executable Code

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”.

VisitorPatternDemo.java

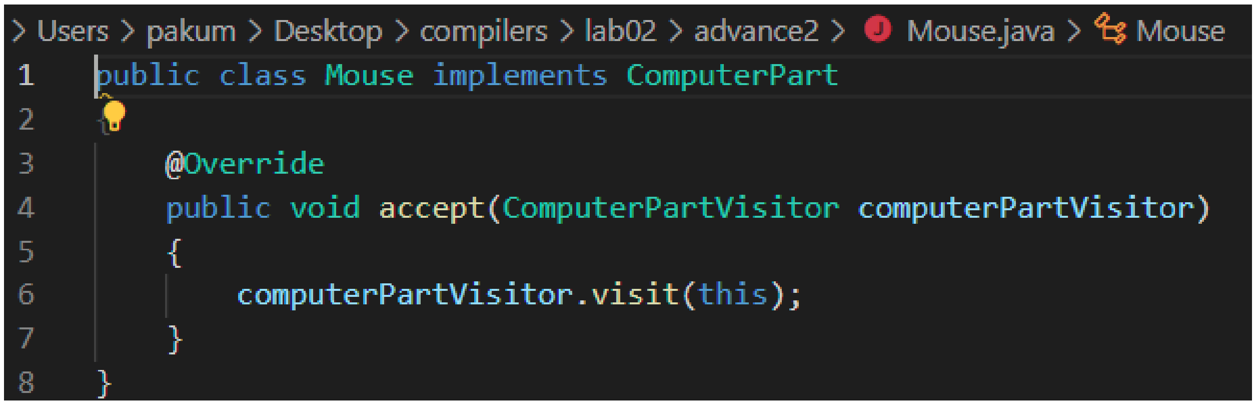


```
> Users > pakum > Desktop > compilers > lab02 > advance2 > VisitorPatternDemo.java
public class VisitorPatternDemo
{
    Run | Debug
    public static void main(String[] args)
    {
        ComputerPart computer = new Computer();
        computer.accept(new ComputerPartDisplayVisitor());
    }
}
```

Computer Components

The code for the following classes is for making components of a computer. The computer is made up of a mouse, keyboard, monitor, and desktop. The computer is the root of the abstract syntax tree.

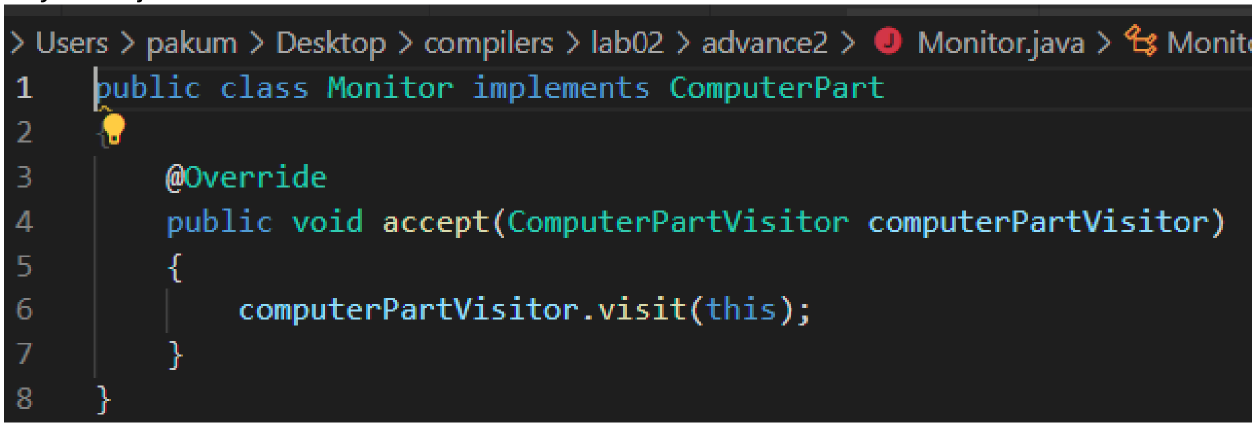
Mouse.java



> Users > pakum > Desktop > compilers > lab02 > advance2 > Mouse.java > Mouse

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

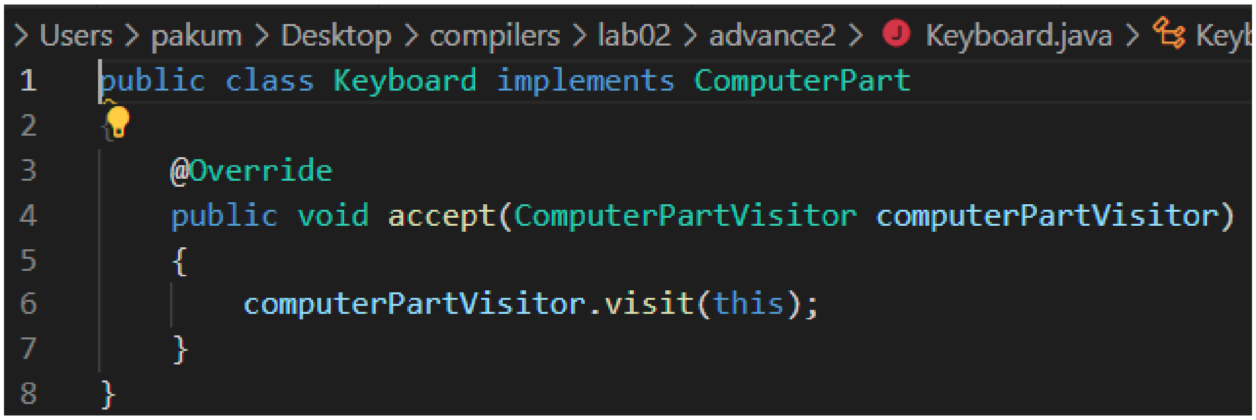
Keyboard.java



> Users > pakum > Desktop > compilers > lab02 > advance2 > Monitor.java > Monit

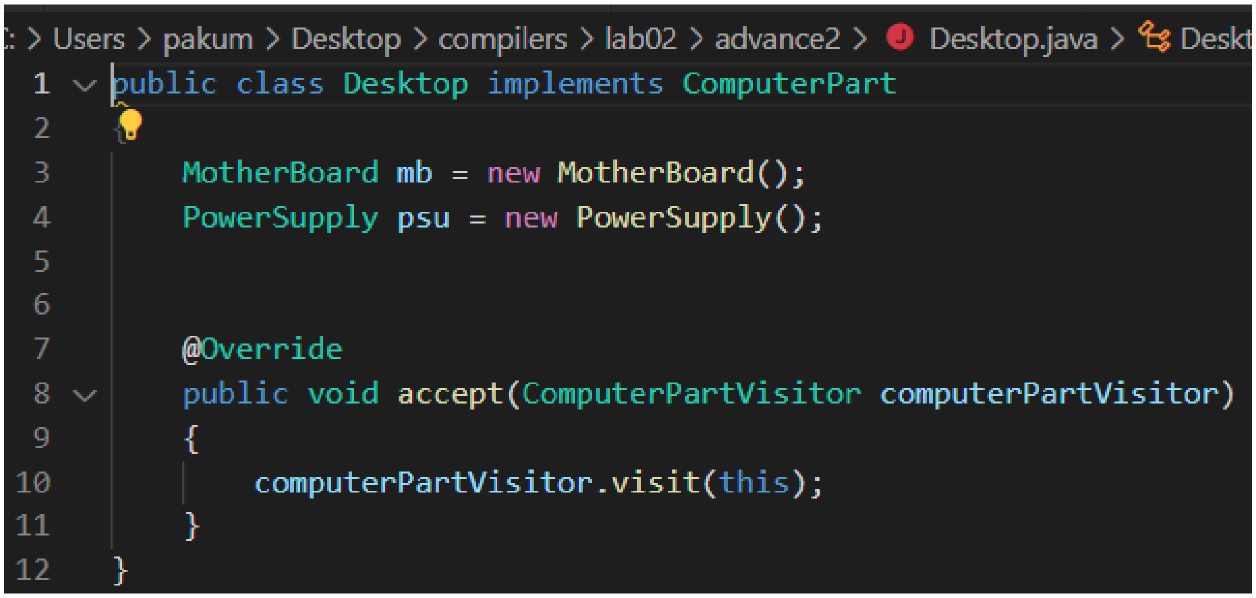
```java
public class Monitor implements ComputerPart
{
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

Monitor.java



```
> Users > pakum > Desktop > compilers > lab02 > advance2 > Keyboard.java > Keyb
public class Keyboard implements ComputerPart
{
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

Desktop.java

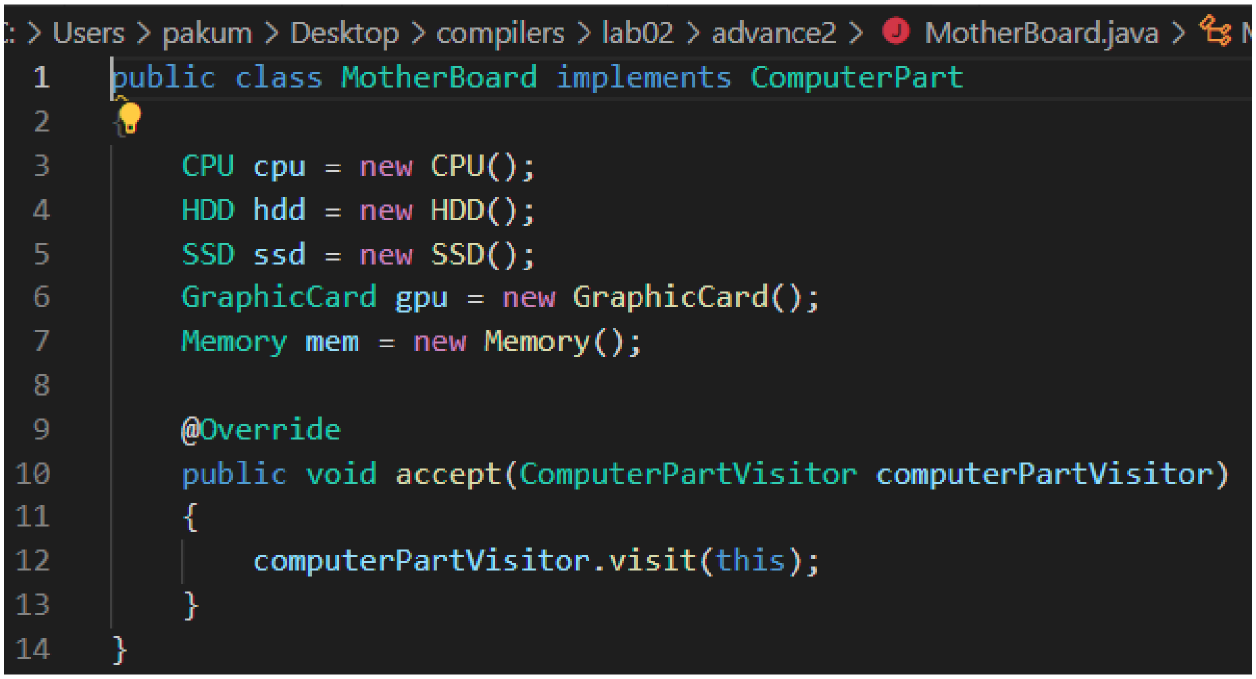


```
: > Users > pakum > Desktop > compilers > lab02 > advance2 > Desktop.java > Deskt
public class Desktop implements ComputerPart
{
    MotherBoard mb = new MotherBoard();
    PowerSupply psu = new PowerSupply();


    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

Desktop Components

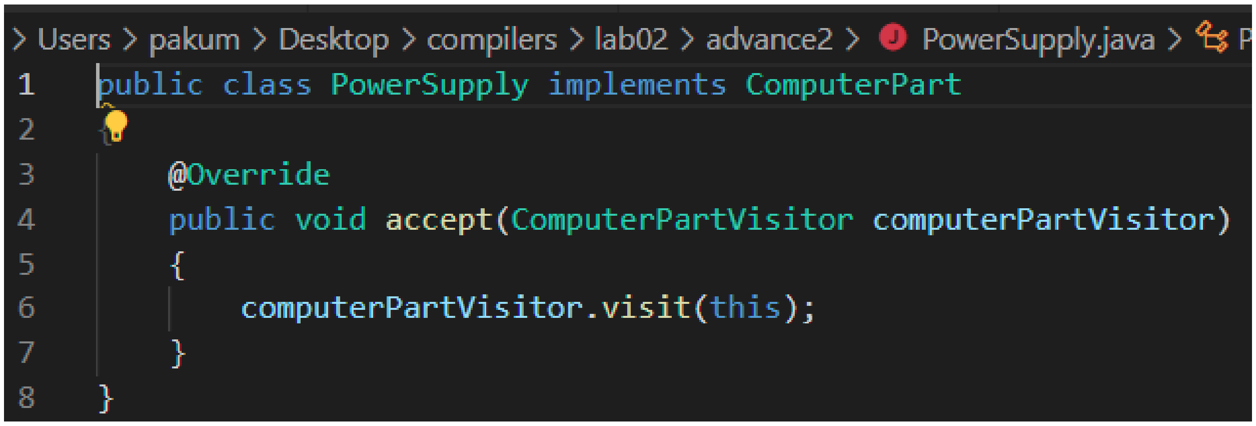
Within the computer there is the Desktop. This desktop contains its own components of a motherboard and a power supply.

MotherBoard.java


: > Users > pakum > Desktop > compilers > lab02 > advance2 > MotherBoard.java >

```java
public class MotherBoard implements ComputerPart
{
    CPU cpu = new CPU();
    HDD hdd = new HDD();
    SSD ssd = new SSD();
    GraphicCard gpu = new GraphicCard();
    Memory mem = new Memory();

    @Override
    public void accept(ComputerPartVisitor computerPartVisitor)
    {
        computerPartVisitor.visit(this);
    }
}
```

PowerSupply.java

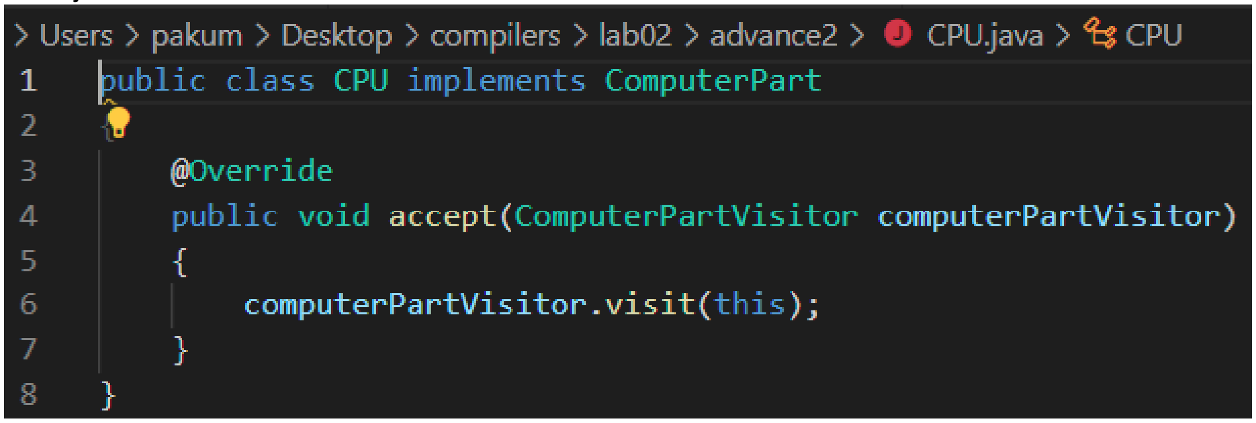

> Users > pakum > Desktop > compilers > lab02 > advance2 > PowerSupply.java >

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

MotherBoard Components

Creating more dept to the abstract syntax tree, the motherboard has its own components. Each component of the motherboard is also a component of a desktop and computer. The components are a cpu, hdd, ssd, gpu, and mem.

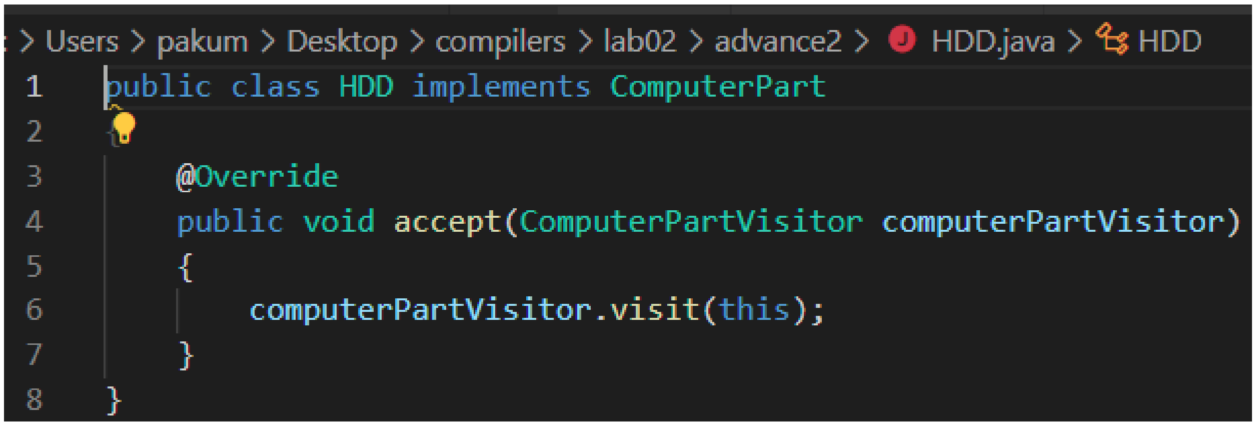
CPU.java



> Users > pakum > Desktop > compilers > lab02 > advance2 > CPU.java > CPU

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

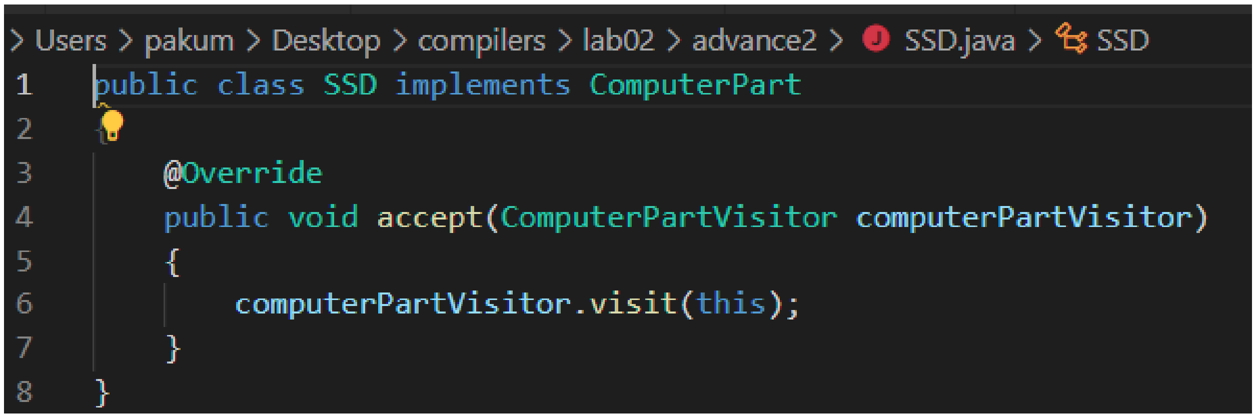
HDD.java



> Users > pakum > Desktop > compilers > lab02 > advance2 > HDD.java > HDD

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

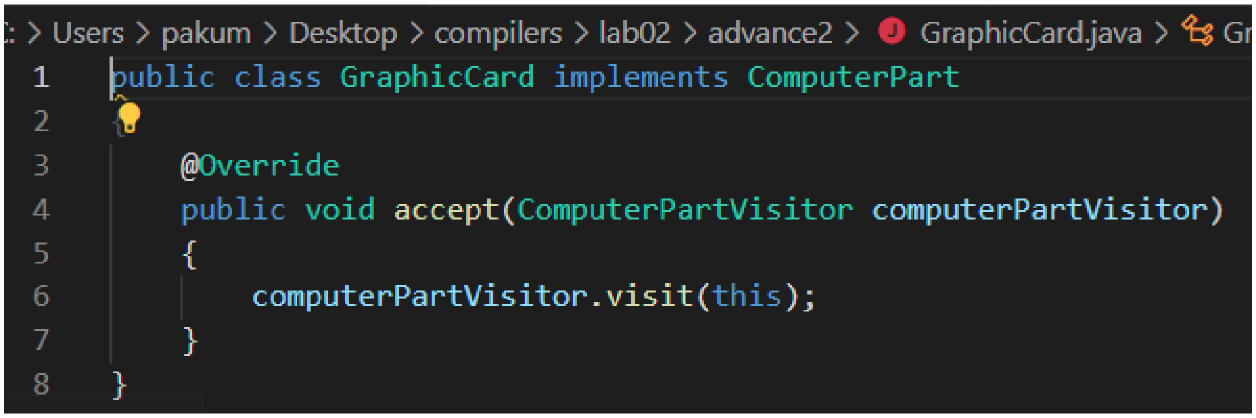
SSD.java



> Users > pakum > Desktop > compilers > lab02 > advance2 > SSD.java > SSD

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

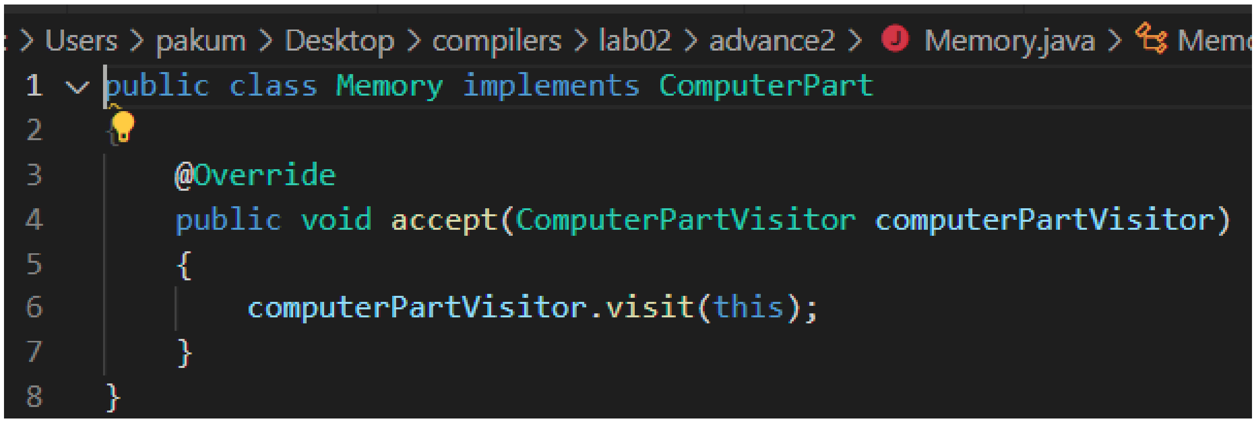
GraphicCard.java



: > Users > pakum > Desktop > compilers > lab02 > advance2 > GraphicCard.java > Gr

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

Memory.java



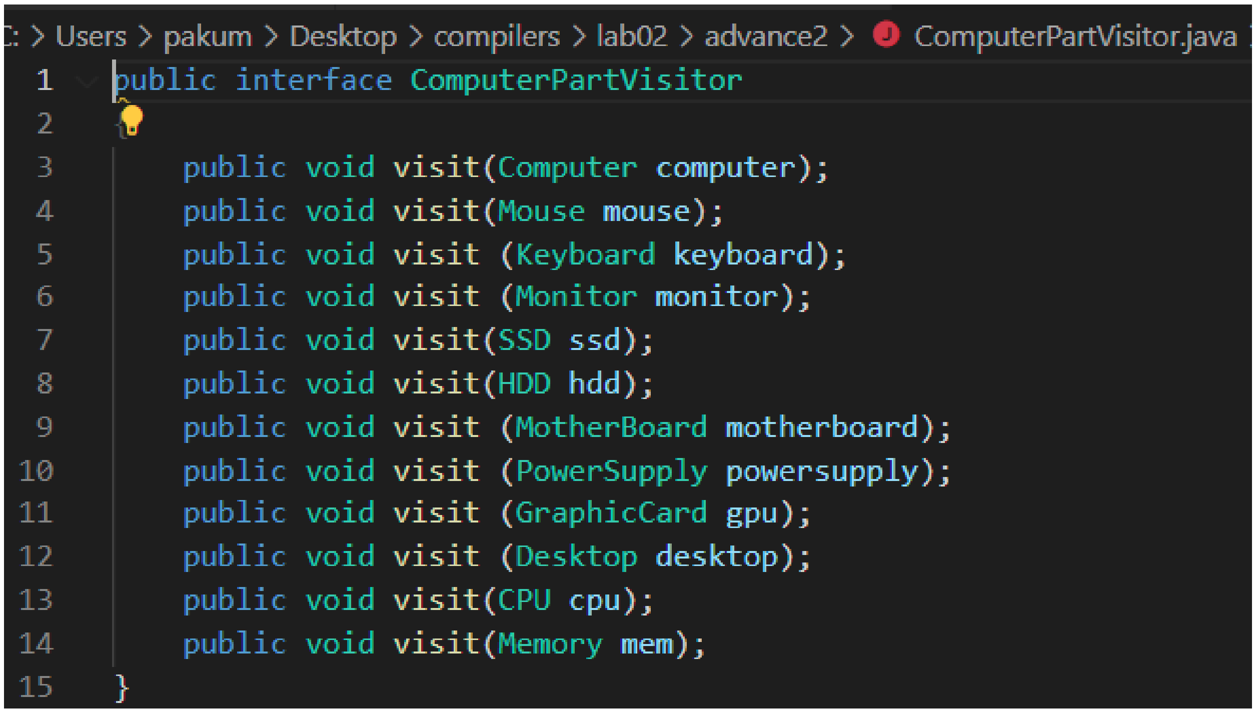
: > Users > pakum > Desktop > compilers > lab02 > advance2 > Memory.java > Meme

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

Using Visitors

For the abstract syntax tree to work, each of the components is encapsulated as a visitor. This visitor is passed into other objects where the methods of visit and accept can be called to get information from the visitor. To make this work, an interface with the visitor must be used for different objects.

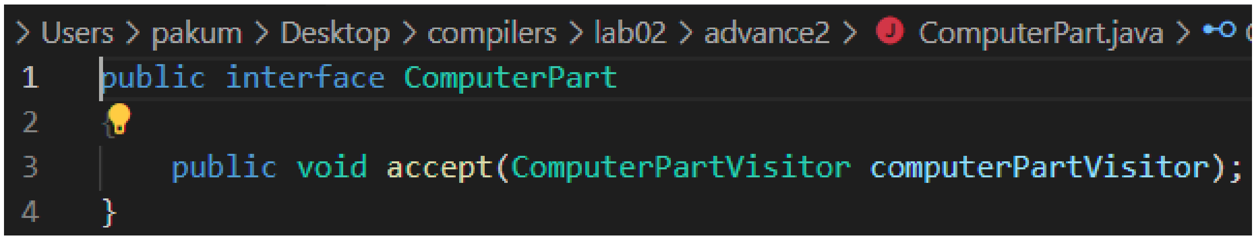
ComputerPartVisitor.java



:\ > Users > pakum > Desktop > compilers > lab02 > advance2 > ComputerPartVisitor.java

```java
public interface ComputerPartVisitor
{
    public void visit(Computer computer);
    public void visit(Mouse mouse);
    public void visit (Keyboard keyboard);
    public void visit (Monitor monitor);
    public void visit(SSD ssd);
    public void visit(HDD hdd);
    public void visit (MotherBoard motherboard);
    public void visit (PowerSupply powersupply);
    public void visit (GraphicCard gpu);
    public void visit (Desktop desktop);
    public void visit(CPU cpu);
    public void visit(Memory mem);
}
```

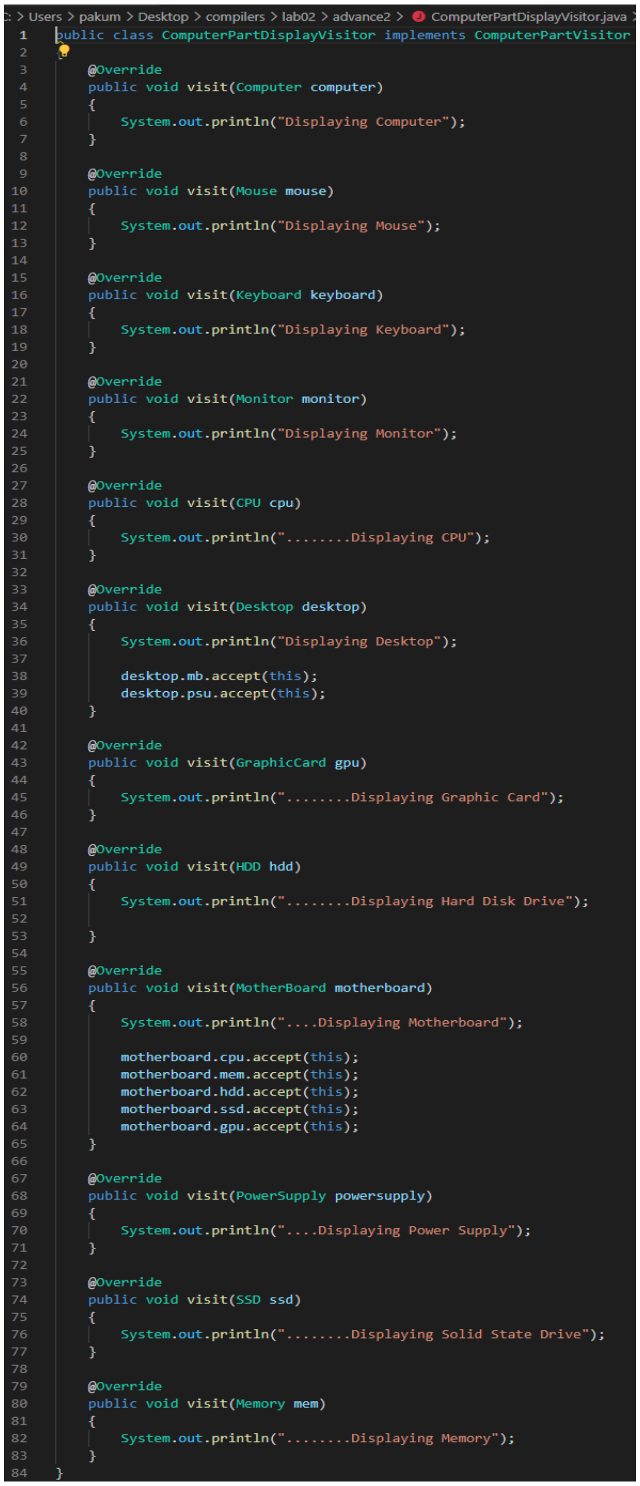
ComputerPart.java



> Users > pakum > Desktop > compilers > lab02 > advance2 > ComputerPart.java >

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

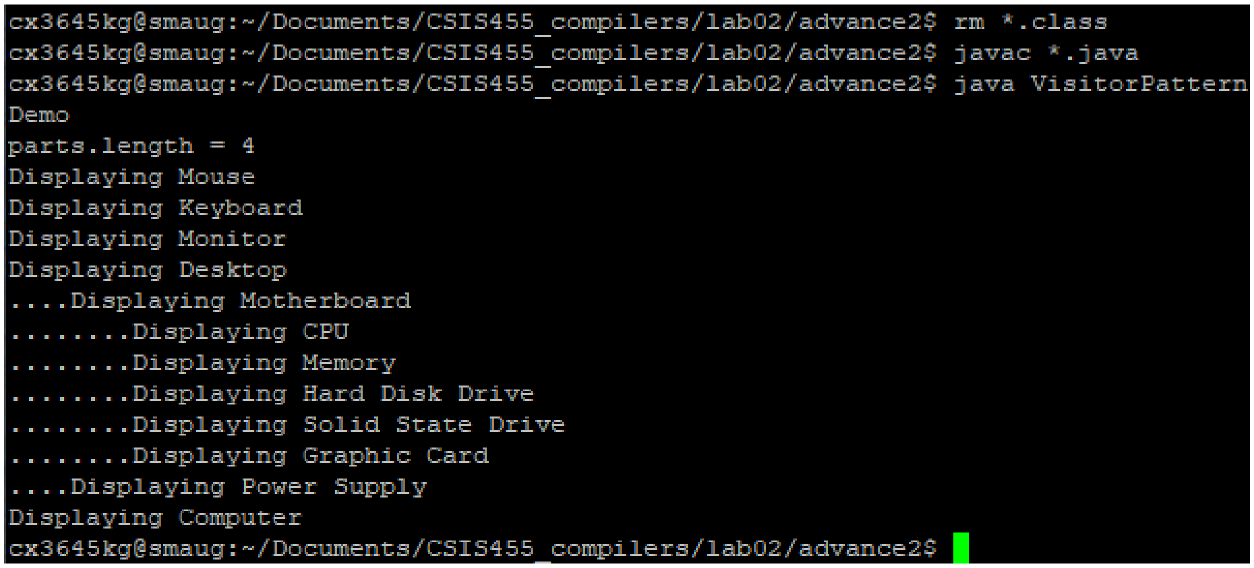
ComputerPartDisplayVisitor.java


: > Users > pakum > Desktop > compilers > lab02 > advance2 > ComputerPartDisplayVisitor.java >

```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");
    }

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

    @Override
    public void visit(Desktop desktop)
    {
        System.out.println("Displaying Desktop");

        desktop.mb.accept(this);
        desktop.psu.accept(this);
    }

    @Override
    public void visit(GraphicCard gpu)
    {
        System.out.println("........Displaying Graphic Card");
    }

    @Override
    public void visit(HDD hdd)
    {
        System.out.println("........Displaying Hard Disk Drive");

    }

    @Override
    public void visit(MotherBoard motherboard)
    {
        System.out.println("....Displaying Motherboard");

        motherboard.cpu.accept(this);
        motherboard.mem.accept(this);
        motherboard.hdd.accept(this);
        motherboard.ssd.accept(this);
        motherboard.gpu.accept(this);
    }

    @Override
    public void visit(PowerSupply powersupply)
    {
        System.out.println("....Displaying Power Supply");
    }

    @Override
    public void visit(SSD ssd)
    {
        System.out.println("........Displaying Solid State Drive");
    }

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

Code Execution

After the java code is compiled, the VisitorPatternDemo can be run. When it runs it will show all the parts of the new computer and roughly show the depth of the AST.



```
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab02/advance2$ rm *.class
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab02/advance2$ javac *.java
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab02/advance2$ java VisitorPattern
Demo
parts.length = 4
Displaying Mouse
Displaying Keyboard
Displaying Monitor
Displaying Desktop
....Displaying Motherboard
........Displaying CPU
........Displaying Memory
........Displaying Hard Disk Drive
........Displaying Solid State Drive
........Displaying Graphic Card
....Displaying Power Supply
Displaying Computer
cx3645kg@smaug:~/Documents/CSIS455_compilers/lab02/advance2$
```

Conclusion

This lab helped solidify some of the concepts of the first lab. It showed how depth can be created in an AST by creating new objects within another object. The root will have no knowledge of what is in the objects farther down the tree, but it can access the parts by visiting and accepting visitors. This helps keep objects contained to whatever is category they fit into. Additionally, the functionality of a visitor is hidden within the object that is currently the visitor. This allows better modularity.

When the code is run, the depth of the tree is shown by visually showing if a component is part of the computer or if it is a component of a component. This also shows that a visitor can traverse a tree to reach all the nodes.