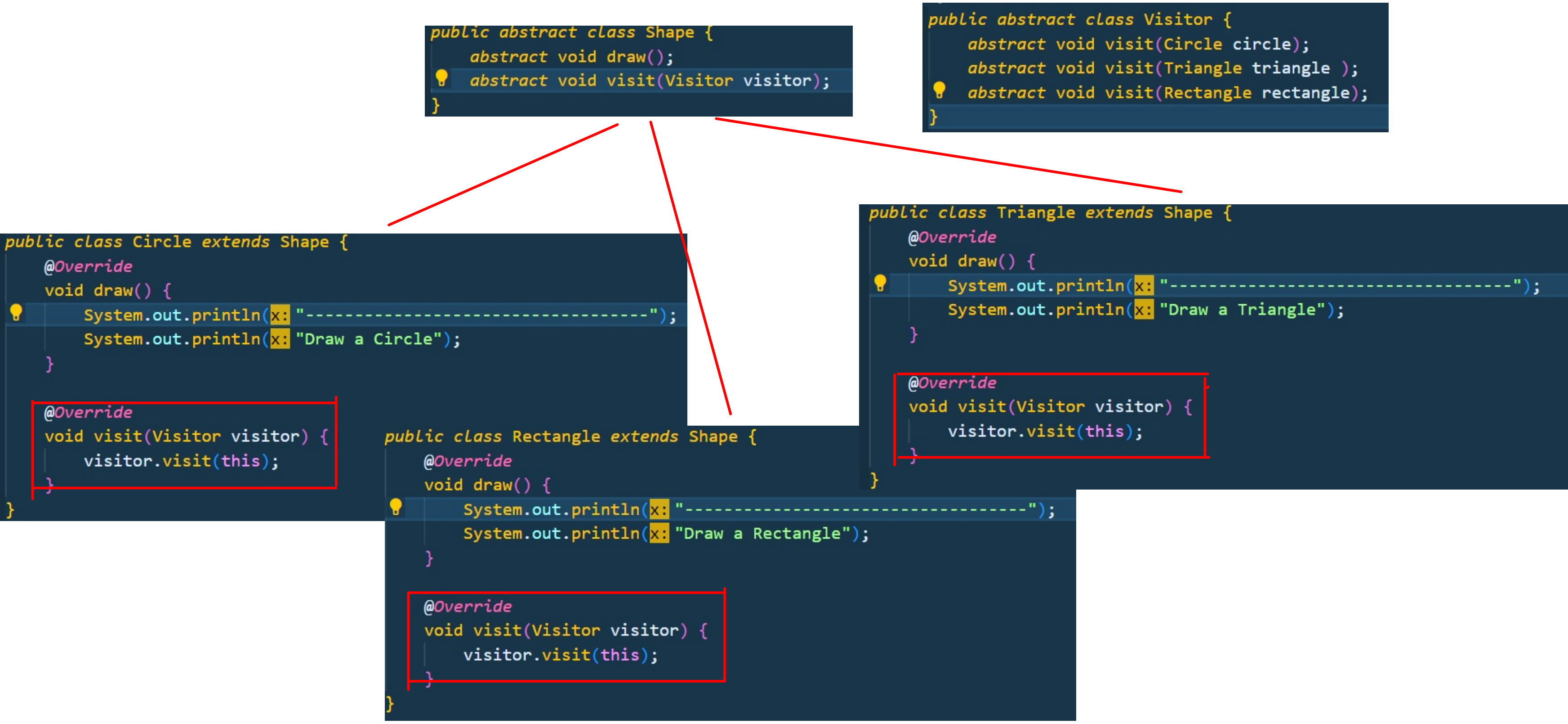
Visitor Design Pattern

This design Pattern adds functionatities to all the collection of classes that are stored with us without actually adding the code for those functionalities in those classes, but writing a visitor which is already added in those classes while production and which calls the visitor's visit function internally and we write the implementation of those visit function in our whole new Visitor functionality Entity. where all the implementation functions for all the classes we have in our Collection.



```java
public abstract class Shape {
    abstract void draw();
    abstract void visit(Visitor visitor);
}
```

```java
public abstract class Visitor {
    abstract void visit(Circle circle);
    abstract void visit(Triangle triangle );
    abstract void visit(Rectangle rectangle);
}
```

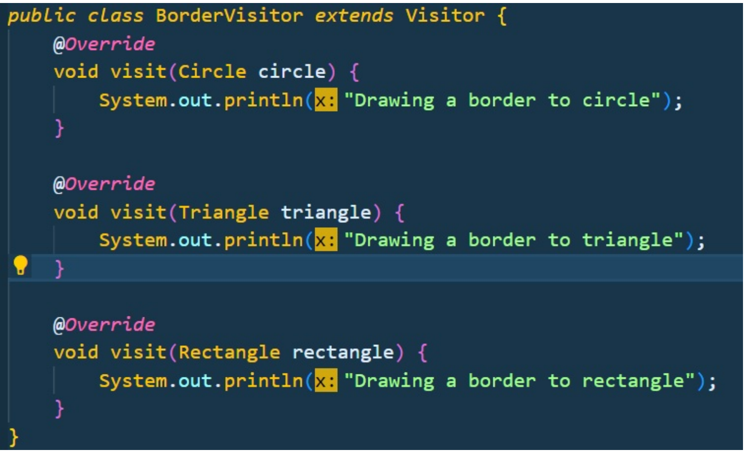
```java
public class Circle extends Shape {
    @Override
    void draw() {
        System.out.println(x: "-----------------------------------");
        System.out.println(x: "Draw a Circle");
    }

    @Override
    void visit(Visitor visitor) {
        visitor.visit(this);
    }
}
```

```java
public class Triangle extends Shape {
    @Override
    void draw() {
        System.out.println(x: "-----------------------------------");
        System.out.println(x: "Draw a Triangle");
    }

    @Override
    void visit(Visitor visitor) {
        visitor.visit(this);
    }
}
```

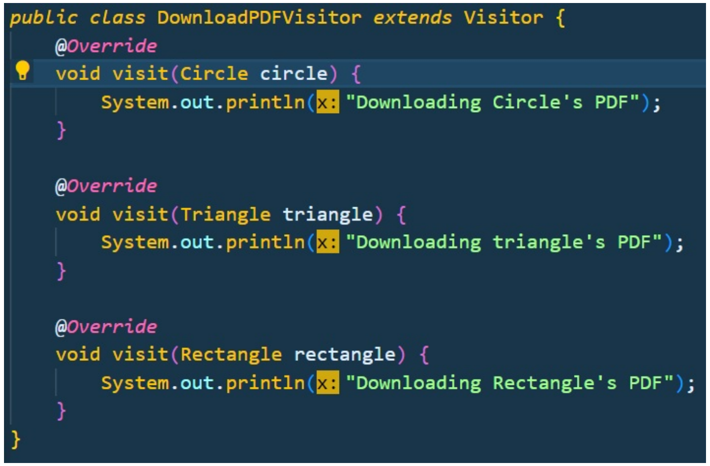
```java
public class Rectangle extends Shape {
    @Override
    void draw() {
        System.out.println(x: "-----------------------------------");
        System.out.println(x: "Draw a Rectangle");
    }

    @Override
    void visit(Visitor visitor) {
        visitor.visit(this);
    }
}
```

Now the Magic Begins!!!!

We write a function Visitor which adds a single functionality implementation of all the collection classes inside it.

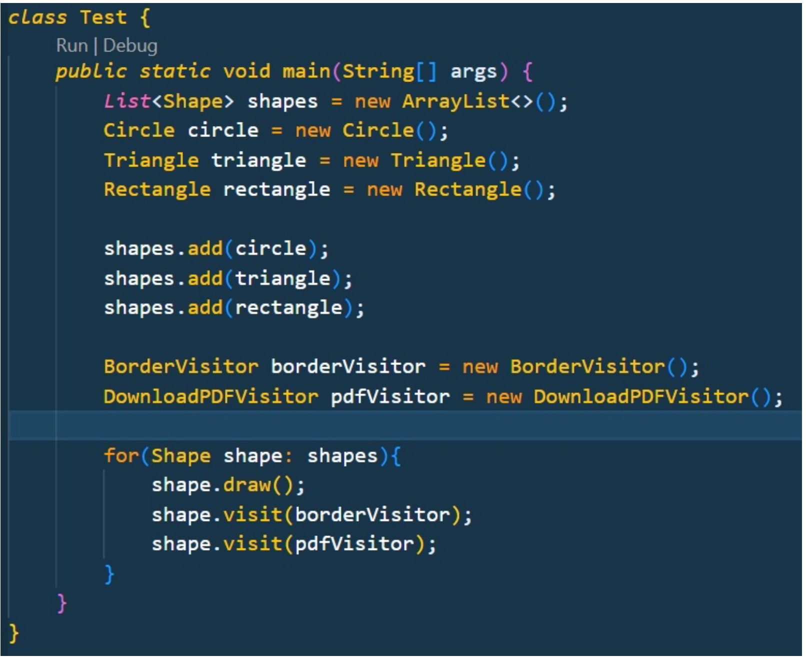


```java
public class BorderVisitor extends Visitor {
    @Override
    void visit(Circle circle) {
        System.out.println(x: "Drawing a border to circle");
    }

    @Override
    void visit(Triangle triangle) {
        System.out.println(x: "Drawing a border to triangle");
    }

    @Override
    void visit(Rectangle rectangle) {
        System.out.println(x: "Drawing a border to rectangle");
    }
}
```

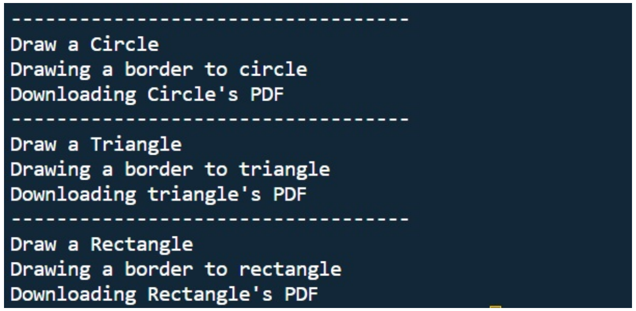


```java
public class DownloadPDFVisitor extends Visitor {
    @Override
    void visit(Circle circle) {
        System.out.println(x: "Downloading Circle's PDF");
    }

    @Override
    void visit(Triangle triangle) {
        System.out.println(x: "Downloading triangle's PDF");
    }

    @Override
    void visit(Rectangle rectangle) {
        System.out.println(x: "Downloading Rectangle's PDF");
    }
}
```

MAGIC.



```
class Test {
    Run | Debug
    public static void main(String[] args) {
        List<Shape> shapes = new ArrayList<>();
        Circle circle = new Circle();
        Triangle triangle = new Triangle();
        Rectangle rectangle = new Rectangle();

        shapes.add(circle);
        shapes.add(triangle);
        shapes.add(rectangle);

        BorderVisitor borderVisitor = new BorderVisitor();
        DownloadPDFVisitor pdfVisitor = new DownloadPDFVisitor();

        for(Shape shape: shapes){
            shape.draw();
            shape.visit(borderVisitor);
            shape.visit(pdfVisitor);
        }
    }
}
```



```
-----------------------------------
Draw a Circle
Drawing a border to circle
Downloading Circle's PDF
-----------------------------------
Draw a Triangle
Drawing a border to triangle
Downloading triangle's PDF
-----------------------------------
Draw a Rectangle
Drawing a border to rectangle
Downloading Rectangle's PDF
```

eg:- circle.draw → Draws a circle.

circle.visit(borderVisitor) {
    bordervisitor. visit (circle)
}

bordervisitor {
    visit (circle) { "Draw a border to circle" }