



---

# Clean Code

“

사용자가 구현을 모른 채  
**자료의 핵심**을  
**조작**할 수 있어야  
진정한 의미의 클래스다.

”

# Chapter 6.

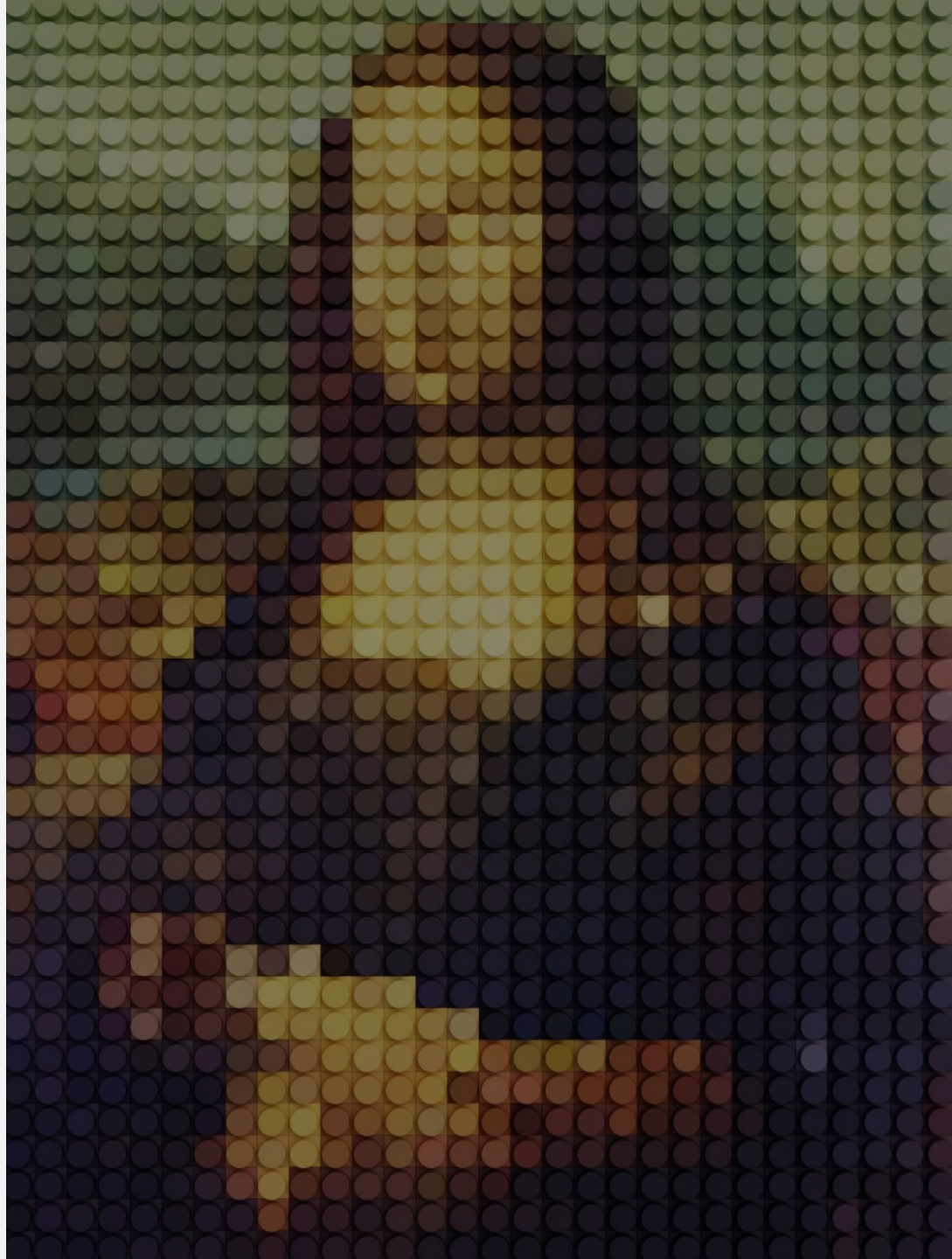
객체와 자료구조

자료 추상화

Q : 프로그래밍에 있어서 추상화는 어떤 의미야?

A : 복잡한 시스템, 데이터 또는  
기능을 **간소화** 하고  
**필요한 부분만 강조**하여  
다루기 쉽게 만드는 개념을 나타냅니다.

- Chat GPT -





# Chapter 6.

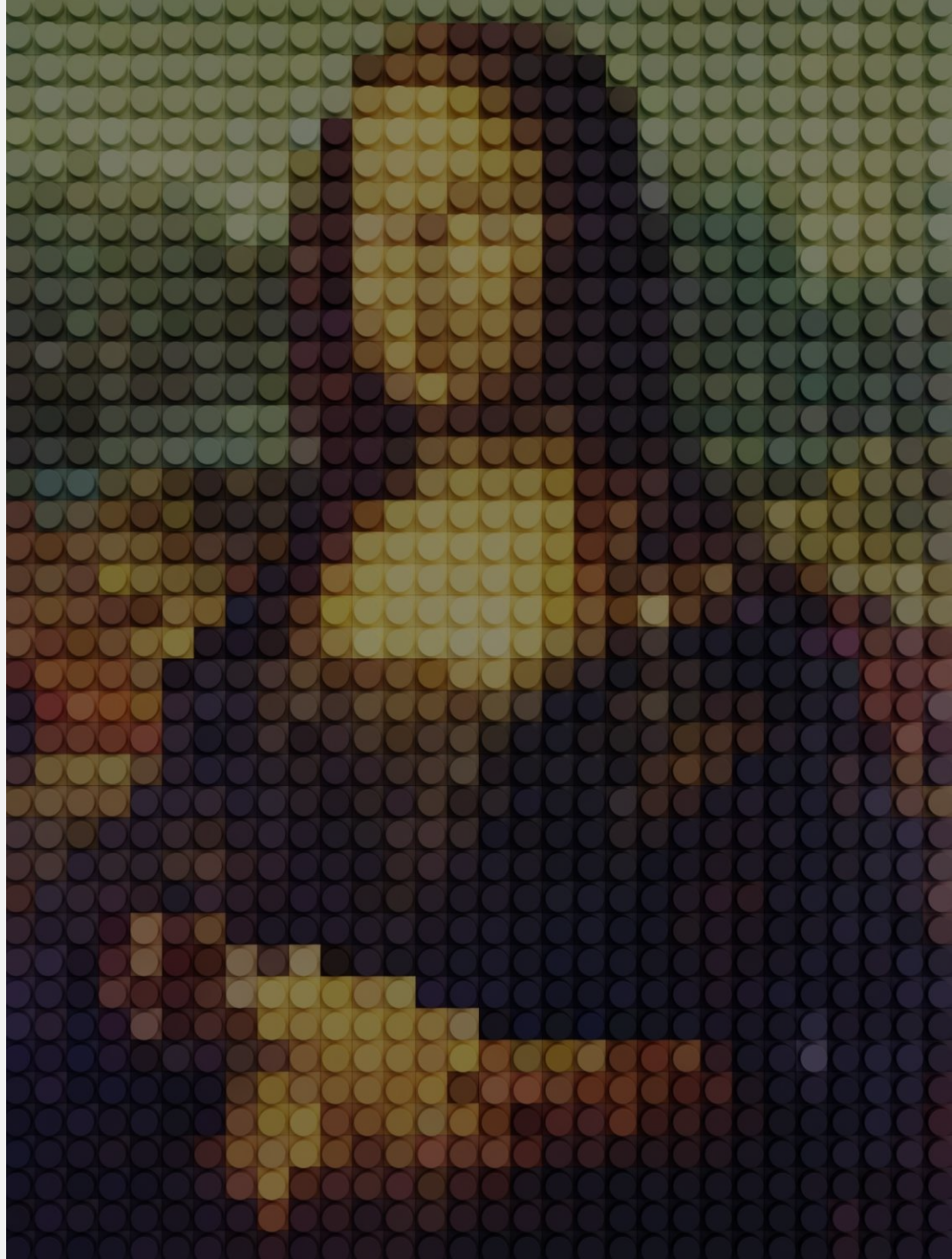
객체와 자료구조

자료 추상화

Q : 프로그래밍에 있어서 추상화는 어떤 의미야?

A : 추상화는 문제를 해결하기 위한  
핵심 아이디어와 개념을 추출하여  
이를 재사용 가능한 모듈로 구성함으로써  
코드의 재사용성과 유지보수성을 향상시킵니다.

- Chat GPT -



# Chapter 6.

자료 추상화

예시코드

Q : 라면 끓이는 방법을 추상화 된 코드로 보여줘

```
java Copy code

public class Main {
    public static void main(String[] args) {
        RamenCooking cooking = new SimpleRamenCooking();
        RamenChef chef = new RamenChef(cooking);
        chef.cookRamen();
    }
}
```

```
java Copy code

public interface RamenCooking {
    void boilWater();
    void prepareNoodles();
    void addSeasoning();
    void serveRamen();
}

public class RamenChef {
    private RamenCooking cooking;

    public RamenChef(RamenCooking cooking) {
        this.cooking = cooking;
    }

    public void cookRamen() {
        cooking.boilWater();
        cooking.prepareNoodles();
        cooking.addSeasoning();
        cooking.serveRamen();
    }
}

public class SimpleRamenCooking implements RamenCooking {
    @Override
    public void boilWater() {
        System.out.println("물을 끓입니다.");
    }

    @Override
    public void prepareNoodles() {
        System.out.println("라면 면을 준비합니다.");
    }

    @Override
    public void addSeasoning() {
        System.out.println("스프와 조미료를 추가합니다.");
    }

    @Override
    public void serveRamen() {
        System.out.println("라면을 제공합니다.");
    }
}
```

# Chapter 6.

자료 추상화

예시코드

Q : 라면 끓이는 방법을 추상화 된 코드로 보여줘

```
java Copy code

public class Main {
    public static void main(String[] args) {
        RamenCooking cooking = new SimpleRamenCooking();
        RamenChef chef = new RamenChef(cooking);
        chef.cookRamen();
    }
}
```

```
java Copy code

public interface RamenCooking {
    void boilWater();
    void prepareNoodles();
    void addSeasoning();
    void serveRamen();
}

public class RamenChef {
    private RamenCooking cooking;

    public RamenChef(RamenCooking cooking) {
        this.cooking = cooking;
    }

    public void cookRamen() {
        cooking.boilWater();
        cooking.prepareNoodles();
        cooking.addSeasoning();
        cooking.serveRamen();
    }
}

public class SimpleRamenCooking implements RamenCooking {
    @Override
    public void boilWater() {
        System.out.println("물을 끓입니다.");
    }

    @Override
    public void prepareNoodles() {
        System.out.println("라면 면을 준비합니다.");
    }

    @Override
    public void addSeasoning() {
        System.out.println("스프와 조미료를 추가합니다.");
    }

    @Override
    public void serveRamen() {
        System.out.println("라면을 제공합니다.");
    }
}
```

# Chapter 6.

자료 추상화

예시코드

Q : 라면 끓이는 방법을 추상화 된 코드로 보여줘

```
java
Copy code

public class Main {
    public static void main(String[] args) {
        RamenCooking cooking = new SimpleRamenCooking();
        RamenChef chef = new RamenChef(cooking);
        chef.cookRamen();
    }
}
```

```
java
Copy code

public interface RamenCooking {
    void boilWater();
    void prepareNoodles();
    void addSeasoning();
    void serveRamen();
}

public class RamenChef {
    private RamenCooking cooking;

    public RamenChef(RamenCooking cooking) {
        this.cooking = cooking;
    }

    public void cookRamen() {
        cooking.boilWater();
        cooking.prepareNoodles();
        cooking.addSeasoning();
        cooking.serveRamen();
    }
}

public class SimpleRamenCooking implements RamenCooking {
    @Override
    public void boilWater() {
        System.out.println("물을 끓입니다.");
    }

    @Override
    public void prepareNoodles() {
        System.out.println("라면 면을 준비합니다.");
    }

    @Override
    public void addSeasoning() {
        System.out.println("스프와 조미료를 추가합니다.");
    }

    @Override
    public void serveRamen() {
        System.out.println("라면을 제공합니다.");
    }
}
```

“

때로는 단순한 자료 구조와  
절차적인 코드가  
가장 적합한 상황도 있다.

”



# Chapter 6.

## 객체와 자료구조

### 자료/객체 비대칭

```
public class Square {
    public Point topLeft;
    public double side;
}

public class Rectangle {
    public Point topLeft;
    public double height;
    public double width;
}

public class Geometry {
    public final double PI = 3.141592653589793;
    public double area(Object shape) throws NoSuchShapeException
    {
        if (shape instanceof Square) {
            Square s = (Square)shape;
            return s.side * s.side;
        }
        else if (shape instanceof Rectangle) {
            Rectangle r = (Rectangle)shape;
            return r.height * r.width;
        }
        else if (shape instanceof Circle) {
            Circle c = (Circle)shape; return PI * c.radius * c.radius;
        }
        throw new NoSuchShapeException();
    }
}
```

```
public class Square implements Shape {
    private Point topLeft;
    private double side;
    public double area() {
        return side*side;
    }
}

public class Rectangle implements Shape {
    private Point topLeft;
    private double height;
    private double width;
    public double area() {
        return height * width;
    }
}

public class Circle implements Shape {
    private Point center;
    private double radius;
    public final double PI = 3.141592653589793;
    public double area() {
        return PI * radius * radius;
    }
}
```