

2025년 상반기 K-디지털 트레이닝

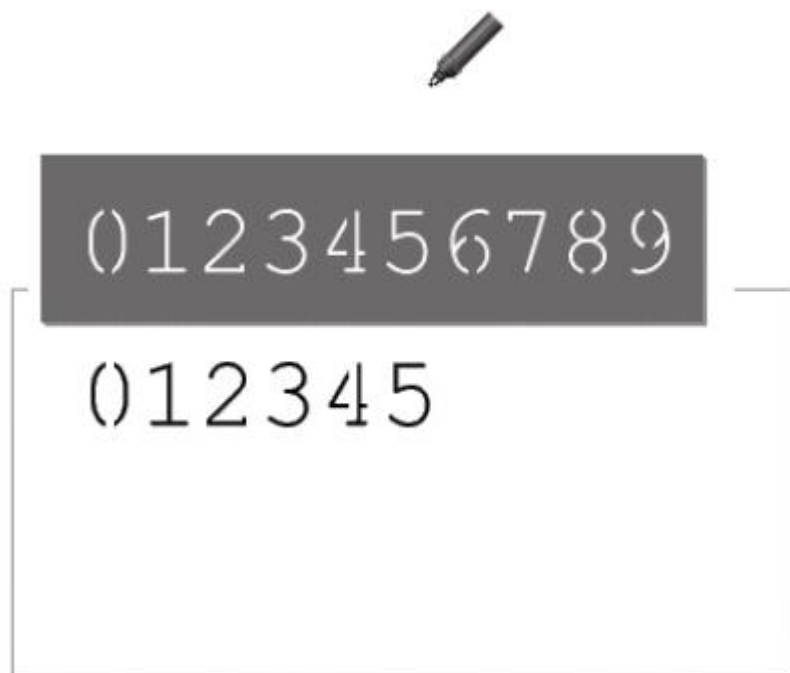
# Template Method

- 하위 클래스에서 구체적으로 처리한다

[KB] IT's Your Life

## ✓ 템플릿

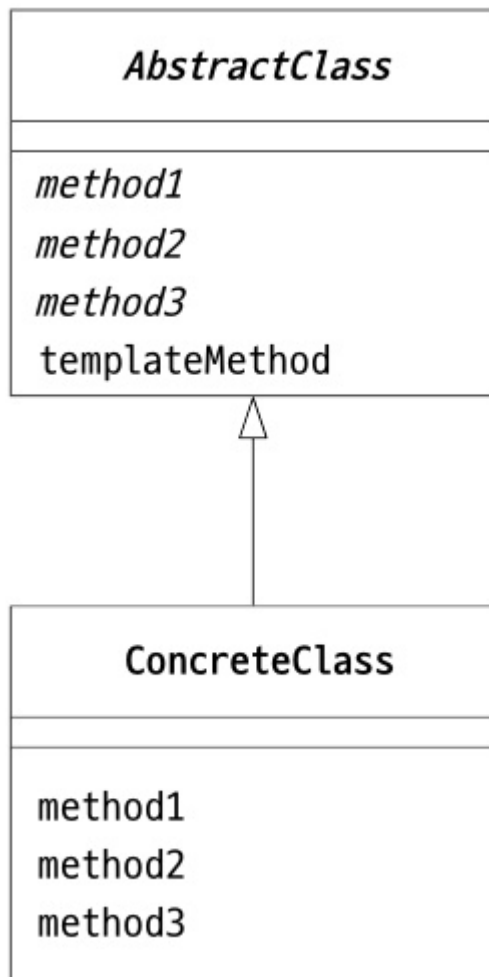
- 문자 모양대로 구멍이 난 얇은 플라스틱 판
  - 템플릿의 구멍을 보면 어떤 형태의 문자인지 알 수 있지만,
  - 실제로 어떤 문자가 될지는 구체적인 필기 도구가 정해지기 전까진 알 수 없음



## ✓ 템플릿 메서드 패턴

- 템플릿 기능을 가진 패턴
  - 상위 클래스 쪽에 템플릿이 될 메소드를 추상 메서드로 정의
  - 상위 클래스의 코드만 보서는 최종적으로 어떻게 처리되는지 알 수 없음
  - 추상 메서드를 호출하는 방법만 알 수 있음
  - 실제로 구현하는 것은 하위 클래스
    - 구체적인 처리 방식이 결정됨
    - 어느 하위 클래스에서 어떻게 구현하더라도 처리의 큰 흐름은 상위 클래스에서 구성한 대로 진행
- 상위 클래스에서 처리의 뼈대를 결정하고 하위 클래스에서 그 구체적인 내용을 결정하는 디자인 패턴

## ✓ 템플릿 메서드 패턴의 클래스 다이어그램



## ✓ 예제 프로그램

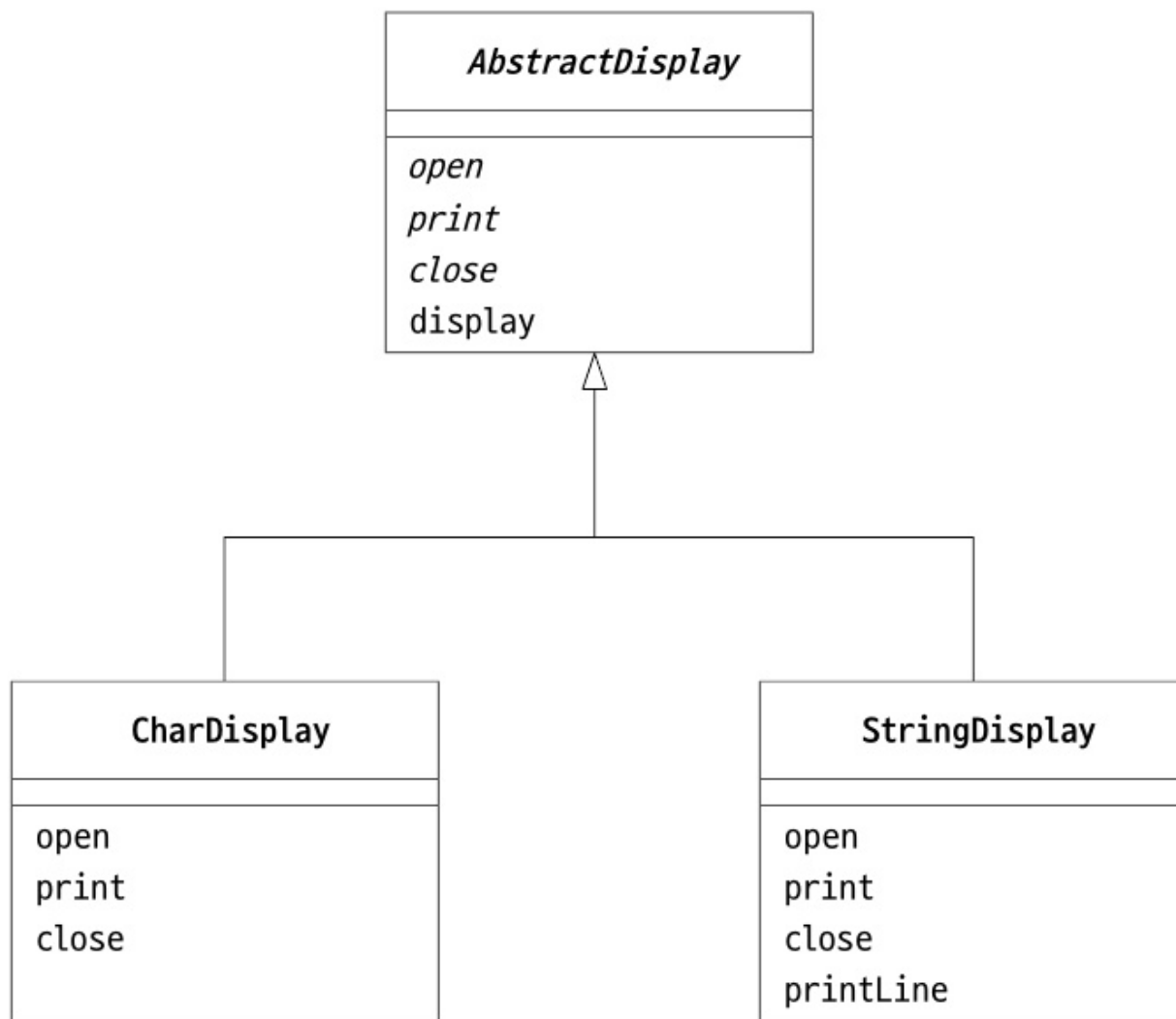
- 문자나 문자열을 5번 반복해서 표시하는 간단한 프로그램

이름	설명
AbstractDisplay	메소드 <code>display</code> 만 구현된 추상 클래스
CharDisplay	메소드 <code>open</code> , <code>print</code> , <code>close</code> 를 구현하는 클래스
StringDisplay	메소드 <code>open</code> , <code>print</code> , <code>close</code> 를 구현하는 클래스
Main	동작 테스트용 클래스

```
<<HHHHH>>
```

```
+-----+
|Hello, world.|
|Hello, world.|
|Hello, world.|
|Hello, world.|
|Hello, world.|
+-----+
```

## ✓ 예제 프로그램의 클래스 다이어그램



## AbstractDisplay.java

```
public abstract class AbstractDisplay {  
    public abstract void open();  
    public abstract void print();  
    public abstract void close();  
  
    // AbstractDisplay에서 구현하는 메서드  
    public final void display() {  
        open();  
        for(int i = 0; i < 5; i++) {  
            print();  
        }  
        close();  
    }  
}
```

## CharDisplay.java

```
public class CharDisplay extends AbstractDisplay{
    private char ch;    // 표시해야 하는 문자

    public CharDisplay(char ch) {
        this.ch = ch;
    }

    @Override
    public void open() {
        System.out.print("<<");
    }

    @Override
    public void print() {
        System.out.print(ch);
    }

    @Override
    public void close() {
        System.out.println(">>");
    }
}
```



## StringDisplay.java

```
public class StringDisplay extends AbstractDisplay{
    private String string; // 표시해야 하는 문자열
    private int width; // 문자열의 길이

    public StringDisplay(String string) {
        this.string = string;
        this.width = string.length();
    }

    @Override
    public void open() {
        printLine();
    }

    @Override
    public void print() {
        System.out.println("|" + string + "|");
    }

    @Override
    public void close() {
        printLine();
    }
}
```

## StringDisplay.java

```
private void printLine() {  
    System.out.print("+");  
    for(int i = 0; i < width; i++) {  
        System.out.print("-");  
    }  
    System.out.println("+");  
}  
}
```

## ✏ Main.java

```
public class Main {  
    public static void main(String[] args) {  
        AbstractDisplay d1 = new CharDisplay('H');  
        AbstractDisplay d2 = new StringDisplay("Hello, world.");  
  
        d1.display();  
        d2.display();  
    }  
}
```

```
<<HHHHH>>  
+-----+  
|Hello, world.|  
|Hello, world.|  
|Hello, world.|  
|Hello, world.|  
|Hello, world.|  
+-----+
```

## ✓ Template Method 패턴을 왜 사용하는가?

- 로직을 공통화할 수 있다
  - 상위 클래스의 템플릿 메소드에 알고리즘이 기술되어 있음
  - 하위 클래스는 알고리즘을 일일이 기술할 필요 없음

→ framework의 기본 패턴

- 상위 클래스와 하위 클래스의 연계 플레이
- 하위 클래스를 상위클래스와 동일시 한다.