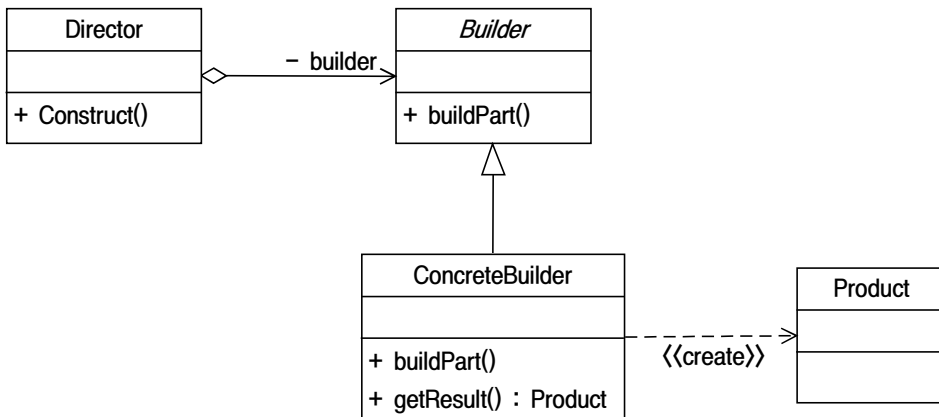


29. 빌더 패턴

빌더 패턴(builder pattern) : 객체 조립 순서에 따라 다른 결과를 생성하는 패턴

〈빌더 패턴 클래스 다이어그램〉



- 빌더 패턴은 복합 객체를 생성하는 방법과 조립하는 방법을 분리하는 것이 목적이다.
- 동일한 객체 생성 절차에서 조립 방법에 따라 다른 결과를 만들 수 있게 하는 패턴이다.
- 즉, 객체 조립 순서에 따라 다른 결과를 만들 수 있게 하는 패턴이다.

Product (생산물)	<ul style="list-style-type: none"> • Product는 생성할 복합 객체를 표현하기 위한 것이다. • Product는 최종적으로 생성될 제품이다.(복합구조)
Builder	<ul style="list-style-type: none"> • Builder는 Product를 생성하기 구성요소를 정의하기 위한 클래스이다. • Builder는 추상클래스 또는 인터페이스로 정의할 수 있다.
ConcreteBuilder	<ul style="list-style-type: none"> • ConcreteBuilder는 Builder에 정의된 추상메서드를 구현한다. • 부품 합성 방식에 따라 여러 구현체를 생성할 수 있도록 한다.
Director	<ul style="list-style-type: none"> • Director는 Builder를 참조하여 객체를 합성한다.(객체 조립) • 객체 조립 순서에 따라 다른 결과가 도출된다. • Director는 Client에 해당한다. • 영어단어 Director는 연출자(감독), 책임자, 구축자라는 의미이다.

- 빌더 패턴은 복합 객체를 생성하는 절차를 좀 더 세밀하게 나눌 수 있다.
- 빌더 패턴은 제품에 대한 내부표현을 다양하게 변화할 수 있다.
- 빌더 패턴은 생성과 표현에 필요한 코드를 분리한다.

예제 1	햄버거를 만드는데 재료를 - 1. 빵, 2. 치즈, 3. 토마토 - 를 사용하는 경우
------	---

〈빌더 패턴 응용프로그램〉

```
class Hamburger { //햄버거(product)
    private int a, b, c; //햄버거 재료 - 1.빵, 2.치즈, 3.토마토
    public Hamburger(int a, int b, int c){ this.a=a; this.b=b; this.c=c;}
    public String toString() { return "[a=" + a + ", b=" + b + ", c=" + c + "]; } //출력 형식
}

interface HamburgerBuilder { //햄버거(product) 생성을 위한 요소 선언
    HamburgerBuilder bed(int a); //빵
    HamburgerBuilder cheese(int b); //치즈
    HamburgerBuilder tomato(int c); //토마토
    Hamburger build(); //햄버거 조립(요리)
}

class ConcreteHamburgerBuilder implements HamburgerBuilder {
    private int a, b, c;
    public HamburgerBuilder bed(int a) { //햄버거에 빵을 사용 - 1번
        this.a = a; return this;
    }
    public HamburgerBuilder cheese(int b) { //햄버거에 치즈 추가 - 2번
        this.b = b; return this;
    }
    public HamburgerBuilder tomato(int c) { //햄버거에 토마토 추가 - 3번
        this.c = c; return this;
    }
    public Hamburger build() { //햄버거 조립(요리)
        Hamburger n = new Hamburger(a, b, c); return n;
    }
}

public class Client{ //햄버거 요리사 - 연출자(Director)
    public static void main(String[] args) {
        HamburgerBuilder hamburgerBuilder = new ConcreteHamburgerBuilder();
        Hamburger h1 = hamburgerBuilder.bed(1).cheese(2).build();
        Hamburger h2 = hamburgerBuilder.bed(1).cheese(2).tomato(3).build();
        System.out.println(h1); //햄버거에 빵, 치즈만 사용(1번, 2번)
        System.out.println(h2); //햄버거에 빵, 치즈, 토마토를 사용(1번, 2번, 3번)
    }
}
```

예제 2	회원가입과 관련된 코드로 집중적으로 객체를 조립하는 것을 보여준다.
------	---------------------------------------

〈빌더 패턴 응용프로그램〉

```

class Member {
    private String name;
    private String address;
    private String hobby;

    public Member(String name, String address, String hobby) { //생성자
        this.name = name; this.address = address; this.hobby = hobby;
    }

    public Member setName(String name) { //회원 이름 설정
        this.name = name; return this;
    }

    public Member setAddress(String address) { //회원 주소 설정
        this.address = address; return this;
    }

    public Member setHobby(String hobby) { //회원 취미 설정
        this.hobby = hobby; return this;
    }

    public void build(){ //회원 이름, 주소, 취미를 빌더해서 출력
        Member m = new Member(name, address, hobby);
        System.out.println(m.name + " : " + m.address + ", " + m.hobby);
    }
}

public class Client {
    public static void main(String[] args) {
        Member member = new Member("이순신", "충무", "말타기"); //초기 빌더 객체 생성
        Member r1 = member.setName("강감찬");
        r1.build(); //빌더 객체에 새로운 값을 설정(조립)
        Member r2 = member.setAddress("서울").setHobby("축구");
        r2.build(); //빌더 객체에 새로운 값을 설정(조립)
        Member r3 = member.setName("홍재연").setAddress("서울").setHobby("야구");
        r3.build(); //빌더 객체에 새로운 값을 설정(조립)
    } //실행결과
} //강감찬 : 충무, 말타기
//강감찬 : 서울, 축구
//홍재연 : 서울, 야구

```
