

2025년 상반기 K-디지털 트레이닝

Builder - 복잡한 인스턴스를 조립한다

[KB] IT's Your Life

✓ Builder 패턴

- 어떤 객체는 new로 인스턴스를 만들었다고 해서 바로 사용할 수 없음
- 일정 절차를 거친 후에 사용할 수 있는 상태가 됨
- 구조를 가진 인스턴스를 만들어 가는 패턴

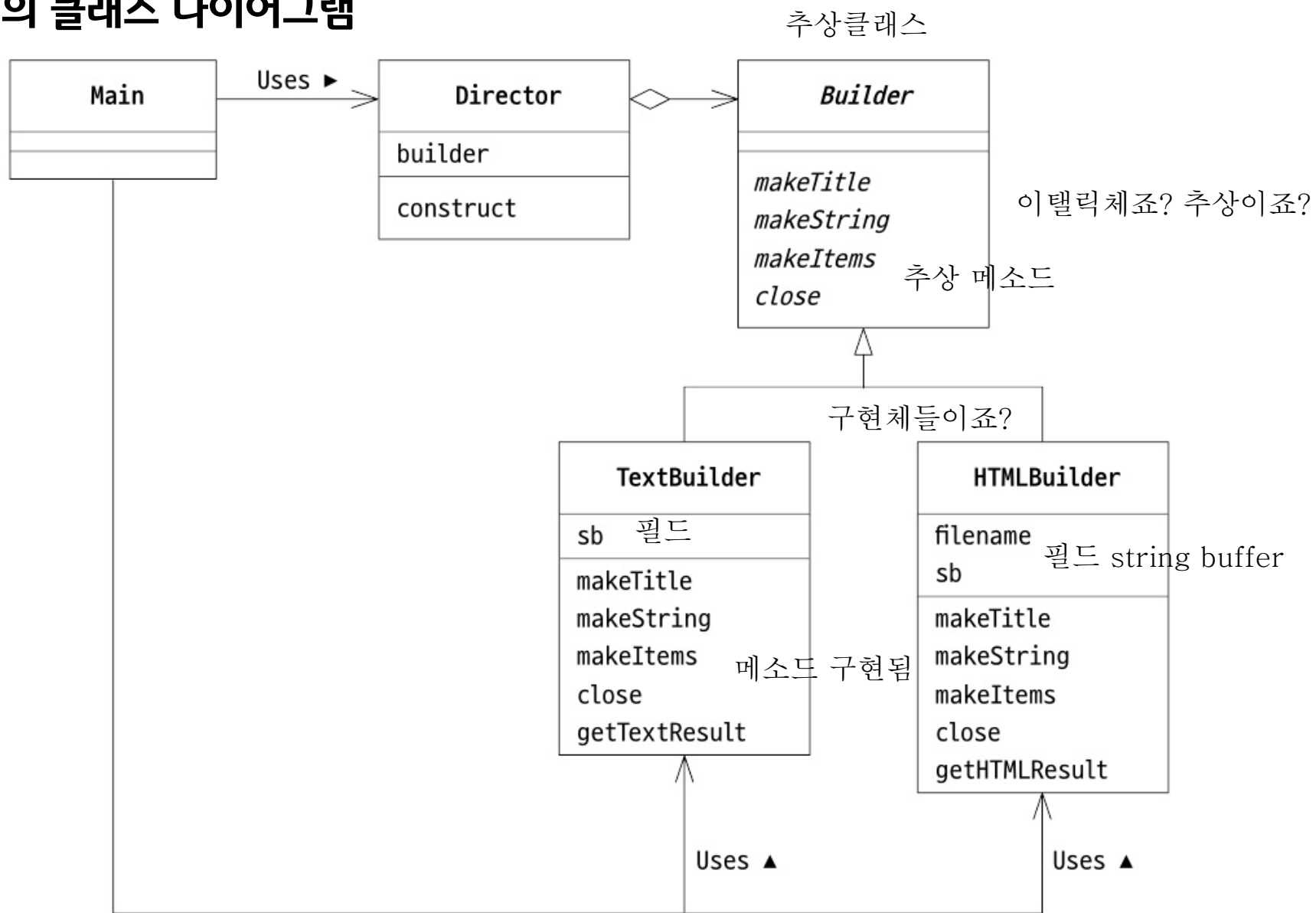
✓ 예제 프로그램

- Builder 패턴을 사용해 '문서'를 작성하는 프로그램
- 여기에서 만들 문서의 구조
 - 타이틀을 한 개 포함
 - 문자열을 몇 개 포함
 - 항목을 몇 개 포함

이름	설명
Builder	문서를 구성하기 위한 메소드를 규정한 추상 클래스 ✓
Director	하나의 문서를 만드는 클래스
TextBuilder	텍스트(일반 문자열)를 이용하여 문서를 만드는 클래스 ✓
HTMLBuilder	HTML 파일을 이용하여 문서를 만드는 클래스
Main	동작 테스트용 클래스

구현 챗들

✓ 예제 프로그램의 클래스 다이어그램



Builder.java

```
public abstract class Builder {  
    public abstract void makeTitle(String title);  
    public abstract void makeString(String str);  
    public abstract void makeItems(String[] items);  
    public abstract void close();  
}
```

Director.java

```
public class Director {
    private Builder builder;

    public Director(Builder builder) {
        this.builder = builder;
    }

    // 문서를 만드는 메서드
    public void construct() {
        builder.makeTitle("Greeting");
        builder.makeString("일반적인 인사");
        builder.makeItems(new String[] {
            "How are you?",
            "Hello.",
            "Hi.",
        });

        builder.makeString("시간대별 인사");
        builder.makeItems(new String[] {
            "Good morning.",
            "Good afternoon.",
            "Good evening.",
        });
        builder.close();
    }
}
```

StringBuilder.java

```
public class StringBuilder extends Builder {
    private StringBuilder sb = new StringBuilder();

    @Override
    public void makeTitle(String title) {
        sb.append("=====\n");
        sb.append("[");
        sb.append(title);
        sb.append("]\n\n");
    }

    @Override
    public void makeString(String str) {
        sb.append("■");
        sb.append(str);
        sb.append("\n\n");
    }
}
```

TextBuilder.java

```
@Override
public void makeItems(String[] items) {
    for(String s: items) {
        sb.append(".");
        sb.append(s);
        sb.append("\n");
    }
    sb.append("\n");
}

@Override
public void close() {
    sb.append("=====\n");
}

public String getTextResult() {
    return sb.toString();
}
}
```


HTMLBuilder.java

```
public class HTMLBuilder extends Builder {
    private String filename = "untitled.html";
    private StringBuilder sb = new StringBuilder();

    @Override
    public void makeTitle(String title) {
        filename = title + ".html";
        sb.append("<!DOCTYPE html>\n");
        sb.append("<html>\n");
        sb.append("<head><title>");
        sb.append(title);
        sb.append("</title></head>\n");
        sb.append("<body>\n");
        sb.append("<h1>");
        sb.append(title);
        sb.append("</h1>\n\n");
    }
}
```

HTMLBuilder.java

```
@Override
public void makeString(String str) {
    sb.append("<p>");
    sb.append(str);
    sb.append("<p>\n\n");
}

@Override
public void makeItems(String[] items) {
    sb.append("<ul>\n");
    for(String s: items) {
        sb.append("<li>");
        sb.append(s);
        sb.append("<li>\n");
    }
    sb.append("</ul>\n\n");
}
```

HTMLBuilder.java

```
@Override
public void close() {
    sb.append("</body>");
    sb.append("</html>\n");
    try(Writer writer = new FileWriter(filename)) {
        writer.write(sb.toString());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public String getHTMLResult() {
    return filename;
}
}
```

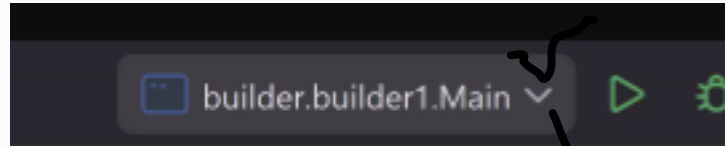
Main.java

```
public class Main {  
  
    // 사용 방법을 표시한다.  
    public static void usage() {  
        System.out.println("Usage: java Main text 텍스트 문서 작성");  
        System.out.println("Usage: java Min html HTML 파일로 문서 작성");  
    }  
}
```

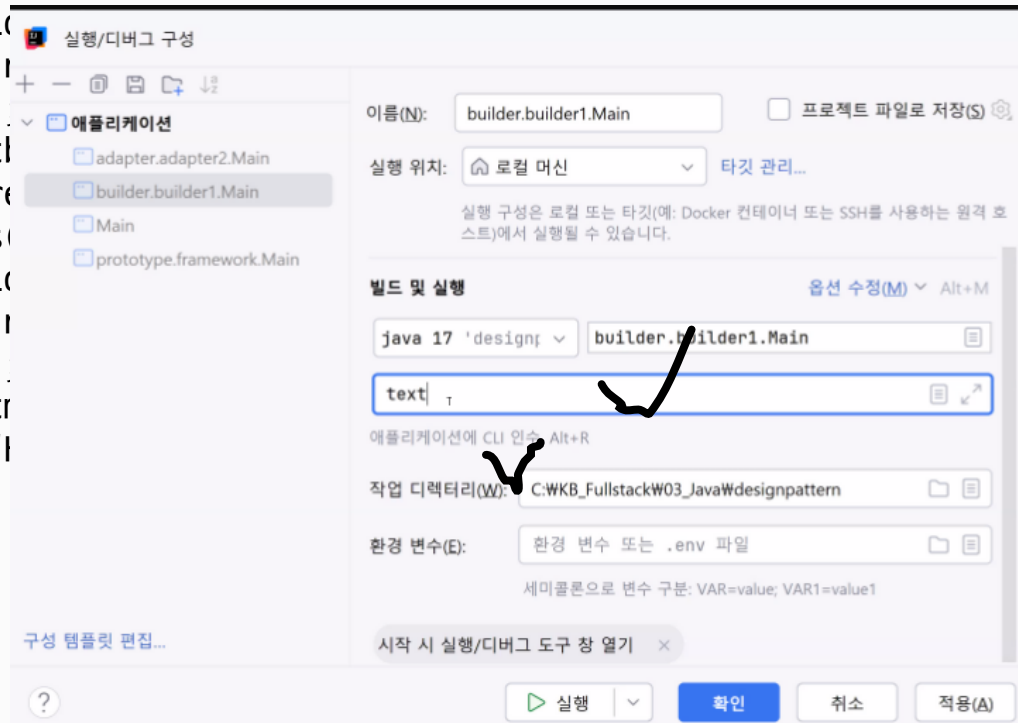
Main.java

```
public static void main(String[] args) {
    if(args.length != 1) {
        usage();
        System.exit(0);
    }
    if(args[0].equals("text")) {
        TextBuilder textbuilder = new TextBuilder();
        Director director = new Director();
        director.construct(textbuilder);
        String result = textbuilder.getResult();
        System.out.println(result);
    } else if(args[0].equals("html")) {
        HTMLBuilder htmlbuilder = new HTMLBuilder();
        Director director = new Director();
        director.construct(htmlbuilder);
        String filename = htmlbuilder.getFilename();
        System.out.println(filename);
    } else {
        usage();
        System.exit(0);
    }
}
```

main argument 실행 구성 설정하기



클릭



=====
[Greeting]

■일반적인 인사

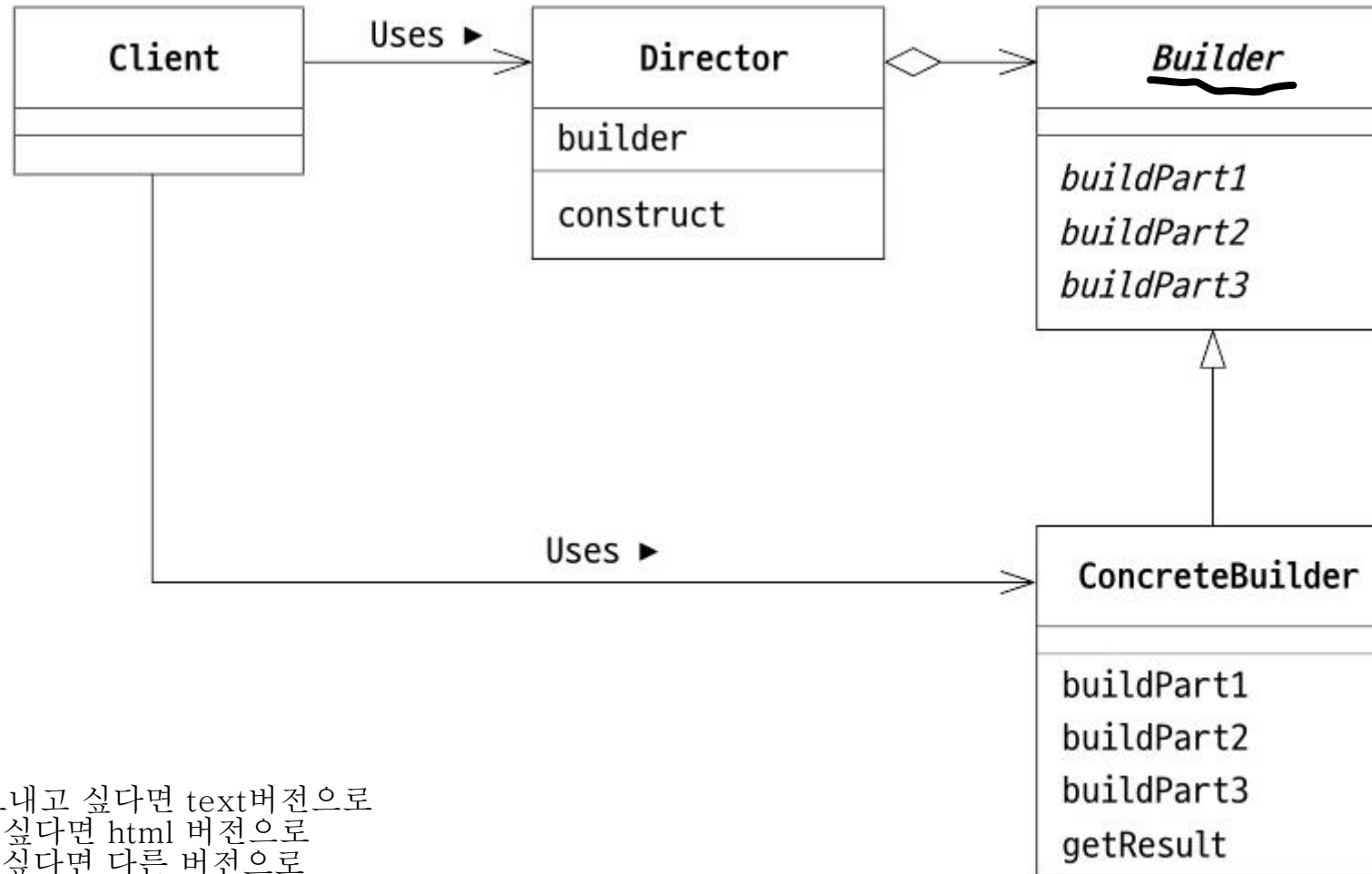
.How are you?
.Hello.
.Hi.

■시간대별 인사

.Good morning.
.Good afternoon.
.Good evening.

=====

✓ Builder 패턴의 클래스 다이어그램

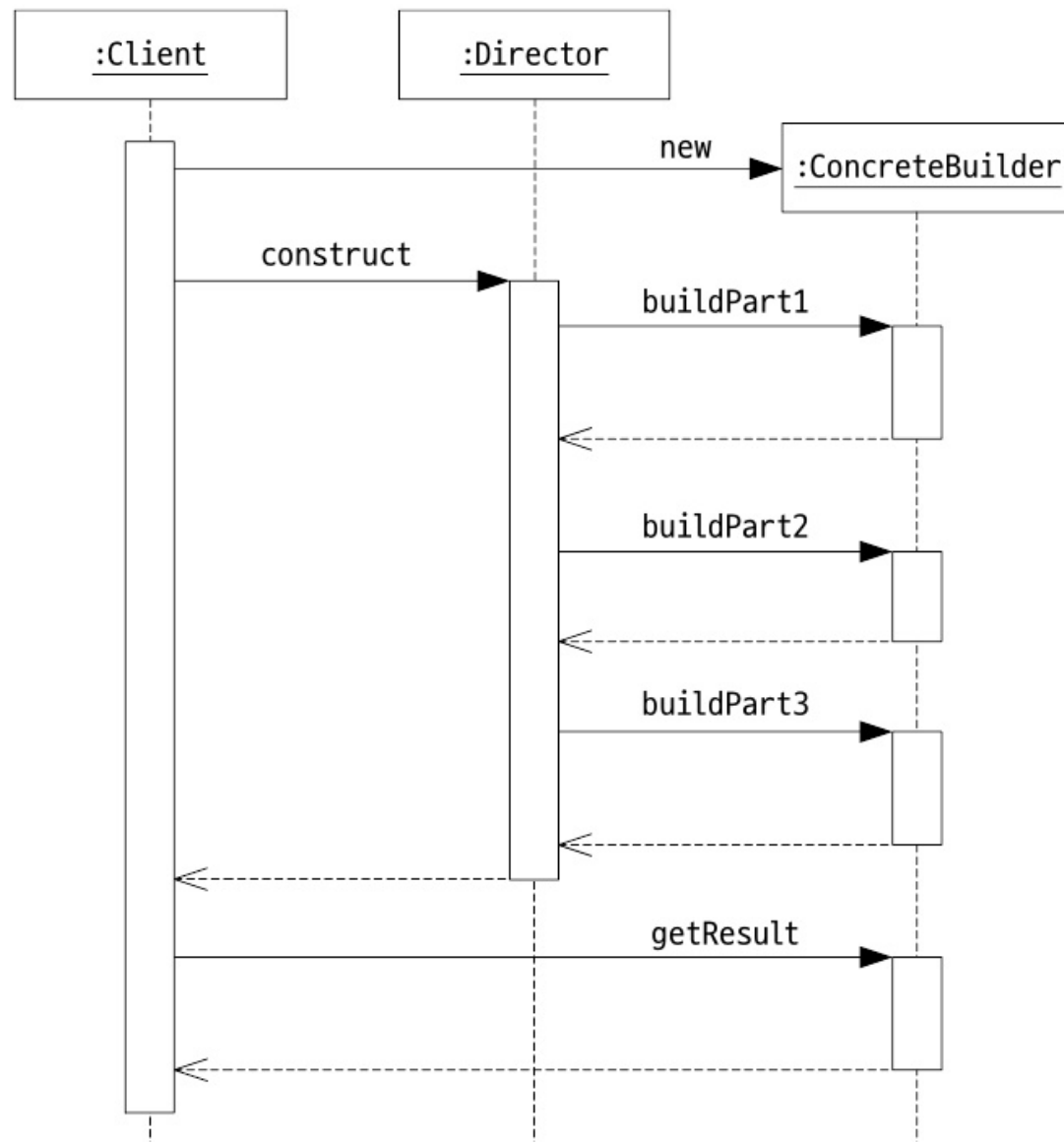


사용성

db에서 mail로 보내고 싶다면 text버전으로
web으로 보내고 싶다면 html 버전으로
.csv으로 보내고 싶다면 다른 버전으로

빌더 패턴을 도입을 하면 매번 다른 형식으로 보내고 싶을 때마다 프로그래밍 할 필요 없다!

✓ Builder 패턴의 시퀀스 다이어그램



✓ 누가 무엇을 알고 있는가?

- 어느 클래스가 어느 메서드를 사용할 수 있는지에 주의하여 프로그래밍 해야함
- Main 클래스
 - Builder 클래스의 메서드를 모름
 - Director 클래스의 construct() 메서드만 호출
- Director 클래스
 - Builder 클래스만 알고 있음
 - 실제로 이용하는 클래스가 무엇인지 알지 못함
 - Builder 클래스의 하위 클래스이면 됨
 - 교체 가능 (OCP)

✓ 예제 프로그램2

- Lombok의 Builder 패턴 스타일

v | |
.
.

@Builder

✏ User.java

```
public class User {
    private String name;
    private String email;
    private String password;
    private String phone;
    private String address;
    private boolean sex;
    private int age;
```

생성자 2개

디폴트 생성자
모든 인자를 받는 생성자

```
public User() {
}
```



@Builder가 위치할 곳. 뒤에 내용들은 명시적으로 어떻게 롬복이 구성하는지에 대한 내용

```
private User(String name, String email, String password, String phone, String address, boolean sex, int age) {
    this.name = name;
    this.email = email;
    this.password = password;
    this.phone = phone;
    this.address = address;
    this.sex = sex;
    this.age = age;
}
```

private으로 했다는 것은 막은 것. 애초에 쓰지 말라고

// Getter, Setter, toString 생략

✏ User.java

static inner class임.

```
public static class Builder {
    private String name;
    private String email;
    private String password; 디폴트 값으로 초기화되긴함
    private String phone;
    private String address;
    private boolean sex;
    private int age;

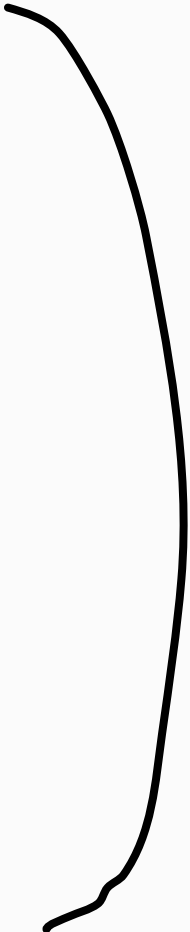
    private Builder() { } // private 생성자이므로 외부에서 생성 불가

    public Builder name(String name) { ✓
        this.name = name;
        return this;
    }

    public Builder email(String email) {
        this.email = email;
        return this;
    }
}
```

✏ User.java

```
public Builder password(String password) {  
    this.password = password;  
    return this;  
}  
  
public Builder phone(String phone) {  
    this.phone = phone;  
    return this;  
}  
  
public Builder address(String address) {  
    this.address = address;  
    return this;  
}  
  
public Builder sex(boolean sex) {  
    this.sex = sex;  
    return this;  
}  
  
public Builder age(int age) {  
    this.age = age;  
    return this;  
}
```



메소드 체이닝을 위해 this를 리턴

✏ User.java

```

public User build() {
    return new User(name, email, password, phone, address, sex, age);
} // end of Builder

public static Builder builder() {
    return new Builder(); // private 생성자 호출
}

```

outer클래스의 private생성자를 사용하여 생성후 반환함 ✓

빌더 자체는 user클래스에서 private builder생성자를 통해서 생성하여 반환함 ✓

Builder b = new User.Builder(); 가능

✏ Main.java

```
public class Main {
    public static void main(String[] args) {
        User user = User.builder()
            .name("홍길동")
            .email("hong@scoula.org")
            .password("123456")
            .phone("010-1111-22222")
            .address("서울시")
            .sex(true)
            .age(16)
            .build();
        System.out.println(user);
    }
}
```

builder 반환됨

메소드 체이닝으로 빌딩

반환된 builder객체의 build 메소드에서 빌딩 완료된 user 반환함.

롬복 @Builder는 어디다가하나

private 생성자 앞에다가 쓰면
해당 생성자에 맞춰서 빌더가 자동으로 써진다.

```
@Entity(name = "users")
@Getter
@ToString
@EqualsAndHashCode(of = "id")
@Builder
@NoArgsConstructor(access = PROTECTED)
@AllArgsConstructor(access = PRIVATE)
```

롬복 어노테이션에서 속성 지정 가능

하지만 보통 롬복 사용시 생성자를 직접 쓰지 않고
노테이션으로 표기하기에 생성자 앞에 명시적으로 @builder
를 하지 못하는데

->

클래스 앞에 @Builder해도 private 멤버에 맞춰서
다해줌.

=> @Data @Builder @NoArgsConstructor @AllArgsConstructor