자바 Record의 사용

Java 14에서 추가된 레코드(Record)에 대해서 레코드의 목적, 자동생성 항목 등 레코드의 기본 사항에 대해 살펴보고 제약사항은 또 어떤 부분이 있는지 알아보도록 하겠습니다.

- 불변(immutable) 데이터 객체를 쉽게 생성할 수 있도록 하는 새로운 유형의 클래스
- JDK14에서 preview로 등장하여 JDK16에서 정식 spec으로 포함

가) 기존의 불변 데이터 객체

```
public class Person {
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

- 상태(name, age)를 보유하는 불변 객체를 생성하기 위한 많은 코드를 작성함
  - 모든 필드에 final을 사용하여 명시적으로 정의
  - 필드 값을 모두 포함한 생성자
  - 모든 필드에 대한 접근자 메서드(getter)
  - 상속을 방지하기 위해 클래스 자체를 final로 선언하기도함
  - 로깅 출력을 제공하기 위한 toString 재정의
  - 두 개의 인스턴스를 비교하기 위한 hashCode, equals 재정의

나) 레코드를 이용한 불변 객체

```
public record Person(String name, int age) {
}
```

• 레코드 클래스를 사용하면 훨씬 간결한 방식으로 동일한 불변 데이터 객체 정의할 수 있음
  - 이름(Person), 헤더(String name, int age), 바디({})

• 컴파일러는 헤더를 통해 내부 필드를 추론
  - 생성자를 만들지 않아도 되고 toString, equals, hashCode 메소드에 대한 구현을 자동으로 제공

▶ 예시 – Person 레코드 클래스의 사용법

```
package ch21;

public class RecordDemo {
    public static void main(String[] args) {
        Person person = new Person("Ted", 20);
        System.out.println("이름 : " + person.name() + ",나이 : " + person.age() );
        System.out.println("객체정보 : " + person.toString());

        Person person1 = new Person("Ted", 20);
        Person person2 = new Person("Dean", 40);

        if(person.equals(person1)){
            System.out.println("person, person1은 같은 사람");
        } else {
            System.out.println("person, person1은 다른 사람");
        }
        if(person.equals(person2)){
            System.out.println("person, person2는 같은 사람");
        } else {
            System.out.println("person, person2는 다른 사람");
        }
    }
}
```

다) 레코드의 장점 및 제한

① 기본적인 동작을 재정의 할 수 있음.

```
public record Person(String name, int age) {
    public Person{
        if(age < 0) {
            throw new IllegalArgumentException("Age cannot be negative");
        }
    }
}
```

② 레코드의 제한

- 레코드는 암묵적으로 final 클래스(상속불가)이고, abstract 선언 불가
- 다른 클래스를 상속(extends) 받을 수 없음, 인터페이스 구현(implements)은 가능

라. Record를 적용한 DTO 설계

※ 공통 사용

▶ TelBookEntity Class

```
public class TelBookEntity {
    private int id;
    private String name;
    private int age;
    private String address;
    private String telNumber;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getTelNumber() {
        return telNumber;
    }

    public void setTelNumber(String telNumber) {
        this.telNumber = telNumber;
    }
}
```

① 적용 전

▶ BeforeTelBookDTO Class

```
public class BeforeTelBookDTO {
	private int id;
	private String name;
	private int age;
	private String address;
	private String telNumber;

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getAge() {
		return age;
	}

	public void setAge(int age) {
		this.age = age;
	}

	public String getAddress() {
		return address;
```

```
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getTelNumber() {
        return telNumber;
    }

    public void setTelNumber(String telNumber) {
        this.telNumber = telNumber;
    }

    public BeforeTelBookDTO(int id, String name, int age, String address, String telNumber) {
        this.id = id;
        this.name = name;
        this.age = age;
        this.address = address;
        this.telNumber = telNumber;
    }

    public static BeforeTelBookDTO of(int id, String name, int age, String address,
                        String telNumber){
        return new BeforeTelBookDTO(id, name, age, address, telNumber);
    }

    public static BeforeTelBookDTO from(TelBookEntity entity){
        return new BeforeTelBookDTO(
                    entity.getId(),
                    entity.getName(),
                    entity.getAge(),
                    entity.getAddress(),
                    entity.getTelNumber()
        );
    }
```

```
    public TelBookEntity toEntity(){
        TelBookEntity entity = new TelBookEntity();
        entity.setId(id);
        entity.setName(name);
        entity.setAge(age);
        entity.setAddress(address);
        entity.setTelNumber(telNumber);
        return  entity;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("id : " + getId() + ", ");
        sb.append("age : " + getAge() + ", ");
        sb.append("address : " + getAddress() + ", ");
        sb.append("tel : " + getTelNumber() + ", ");
        return sb.toString();
    }
}
```

▶ TestBeforeMain Class

```
public class TestBeforeMain {
    public static void main(String[] args) {

        int id = 1;
        String name = "Kim";
        int age = 20;
        String address = "서울";
        String tel = "010-1111-2222";

        BeforeTelBookDTO dto1 = BeforeTelBookDTO.of(id, name, age, address, tel);
        System.out.println(dto1);
```

```
        TelBookEntity entity = new TelBookEntity();
        entity.setId(id);
        entity.setName(name);
        entity.setAge(age);
        entity.setAddress(address);
        entity.setTelNumber(tel);

        BeforeTelBookDTO dto2 = BeforeTelBookDTO.from(entity);
        System.out.println(dto2);
    }
}
```

② Record 적용 후

▶ AfterTelBookDTO Class

```
public record AfterTelBookDTO(
        int id,
        String name,
        int age,
        String address,
        String telNumber
) {
    public static AfterTelBookDTO of(int id, String name, int age, String address,
                                     String telNumber){
        return new AfterTelBookDTO(id, name, age, address, telNumber);
    }

    public  static AfterTelBookDTO from(TelBookEntity entity){
        return AfterTelBookDTO.of(
                entity.getId(),
                entity.getName(),
                entity.getAge(),
                entity.getAddress(),
                entity.getTelNumber());
    }
```

```
    public TelBookEntity toEntity(){
        TelBookEntity entity = new TelBookEntity();
        entity.setId(id);
        entity.setName(name);
        entity.setAge(age);
        entity.setAddress(address);
        entity.setTelNumber(telNumber);

        return  entity;
    }
}
```

▶ TestAfterMain Class

```
public class TestAfterMain {
    public static void main(String[] args) {
        int id = 2;
        String name = "Park";
        int age = 30;
        String address = "부산";
        String tel = "010-3333-5555";

        AfterTelBookDTO dto1 = AfterTelBookDTO.of(id, name, age, address, tel);
        System.out.println(dto1);

        TelBookEntity entity = new TelBookEntity();
        entity.setId(id);
        entity.setName(name);
        entity.setAge(age);
        entity.setAddress(address);
        entity.setTelNumber(tel);

        AfterTelBookDTO dto2 = AfterTelBookDTO.from(entity);
        System.out.println(dto2);
    }
}
```