자바 저장 빈 클래스 생성 문법

1. 빈 클래스 생성 문법

형식)

```
 pubic class 클래스명{

    private String name;
     중략
    //빈클래스는 변수명을 정의할 때 반드시 private으로 정의한다. 이유는 내 자신
   //클래스내에서만 접근 할 수 있기 때문이다. 데이터 보안성

    public void setName(String name){//setter() 메서드 정의(값을 저장)

      this.name=name;
//좌측 멤버변수와 우측 매개변수명이 같을때는 멤버변수 앞에 this.을 붙여야 값이
//저장된다. 만약 this.을 생략하면 값이 저장 안된다.
//this는 내 자신 인스턴스 가리키는 참조 변수이다.
    }
    public String getName(){//getter() 메서드 정의
      return name;//return 키워드로 값을 반환
    }



}
```

2. 빈클래스명을 사용하는 이유는 중간에 자료를 저장하는 역할을 하기 위해서 이다.

3. 빈클래스는 자바의 기본타입 8개에 포함 안되는 레퍼런스(참조) 타입인 클래스형에 해당한다.

클래스(정적) 메서드와 클래스(정적) 변수

1. static 예약어로 정의된 메서드를 클래스(정적)메서드라 한다. 클래스 메서드는 객체 생성 없이 클래스로 직접 접근 한다. 물론 new키워드로 생성된 모든 객체에 의해서 정적메서드는 공유되기 때문에 생성된 객체로도 접근할 수 있다.

   클래스 메서드 내에 인스턴스 변수와 this를 사용할 수 없다.

2. staic 예약어로 정의된 변수를 클래스(정적)변수라 한다. 정적변수도 클래스로 직접 접근 한다.

자바의 **메서드 오버로딩** 개념

1. **메서드 오버로딩**이란 **한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것을 말한다.**
2. **메서드 오버로딩 구분 요건**
   첫째, **메서드의 전달인자 개수를 다르게 한다.**
   둘째, **메서드의 전달인자 타입을 다르게 한다.**
   셋째, **메서드의 전달인자 순서를 다르게 한다.**

자바의 **생성자** 특징

1. 생성자 이름은 **클래스 이름과 같다.**
2. 생성자는 **new 클래스명(); 에 의해서 호출**된다.
3. 생성자는 메서드의 일종이다.
4. 생성자의 **리턴 타입이 없다.**
5. 생성자는 **객체가 생성될 때 호출되는 멤버 변수 초기화 메서드**이다.
6. 생성자는 **오버로딩이 가능**하다.
7. **같은 클래스에서 다른 생성자 호출 법**
   예) this();
8. **매개변수가 없는 생성자를 기본(default) 생성자**라 한다. **기본 생성자**는 **자바 컴파일러가 묵시적으로 제공하기 때문에 코드상에서 생략가능**하다. 하지만 **생성자가 오버로딩이 된 경우는 묵시적 제공을 하지 않는다**. 그러므로 **오버로딩이 된 경우는 기본 생성자를 명시적인 코드를 하는 습관을 가지는 경우가 좋다**.

레코드 선언

데이터 전달을 위한 DTO(Data Transfer Object)를 작성할 때 반복적으로 사용되는 코드를 줄이기 위해 Java 14버전부터 레코드(record)가 도입되었다. 예를 들어 사람의 정보를 전달하기 위한 Person DTO가 있다고 하자.

```
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 name(){return name;}
  public int age(){ return this.age;}

  @Override
  public int hashCode(){ ..}

  @Override
  public boolean equals(Object obj){..}

  @Override
  public String toString(){ ..}
}
```

Person 의 데이터 필드는 읽기만 가능하도록 필드를 private final로 선언하고, 필드 이름과 동일한 Getter() 메서드(name(), age())를 가지고 있다. 그리고 동등비교를 위해 hashCode(). equals() 메서드를 오버라이딩하고 있다. 문자열 출력을 위해 toString() 메서드를 오버라이딩을 했다.

다음 코드는 위와 동일한 코드를 생성하는 레코드 선언이다. class 키워드 대신 record로 바꾸고 클래스 이름 뒤에 괄호를 작성해서 저장할 데이터 종류를 변수로 선언하였다.

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

위와 같이 선언하고 레코드 소스를 컴파일 하면 변수의 타입과 이름을 이용해서 private final 필드가 자동으로 생성되고, 생성자 및 필드 이름과 동일한 Getter() 메서드(name(), age())가 자동으로 추가된다. 그리고
hashCode(), equals(), toString() 메서드가 오버라이딩 되면서 코드로 자동 추가된다.

```
package 새로운문법;
```

```
record Member(String id,String name,int age) {//자바 14에서 추가된 레코드타입 선언
/*레코드는 불변의 값이다. 생성자를 통한 레코드가 생성될 때 전달받은 매개값을 필드값으로 셋팅되고, 사용중 일때는 필드값을 변경할 수 없다.
*/
}
public class RecordExample {
	public static void main(String[] args) {

		Member m = new Member("winter","눈송이", 25);//생성자 자동추가

		System.out.println(m.id());//필드명과 같은 getter()메서드 자동추가
		System.out.println(m.name());
		System.out.println(m.age());
		System.out.println(m.toString());//문자열 출력을 위한 toString()메서드 오버라이딩
		System.out.println();

		Member m1=new Member("spring","개나리",26);
		Member m2=new Member("spring","개나리",26);
		System.out.println("m1.equals(m2) : " + m1.equals(m2));//동등비교 equals()메서드 오버라이딩 자동추가
	}
}
```

롬복 사용하기

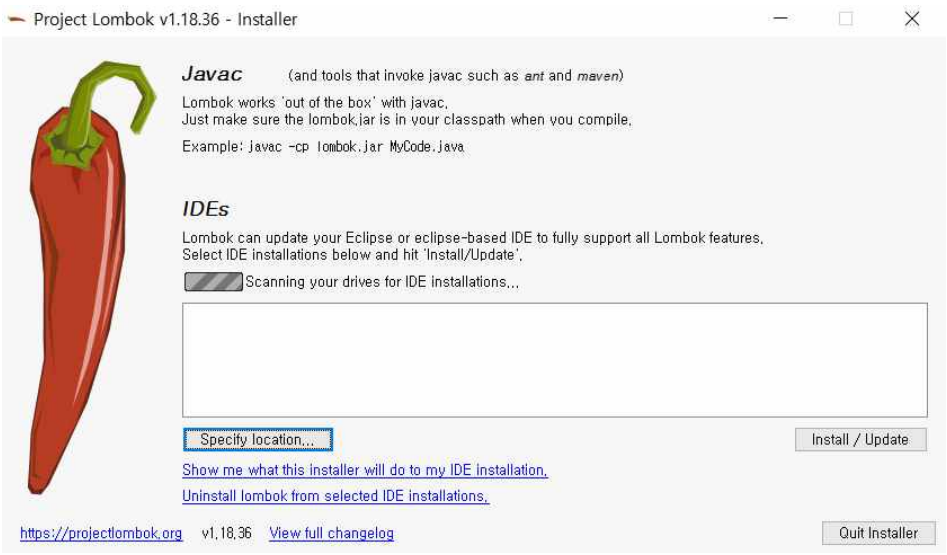
롬복(lombok)은 DTO 클래스를 작성할 때 setter(), getter(), hashCode(), equals(), toString() 메서드등을 자동으로 생성하기 때문에 코드양을 줄일 수 있다. 레코드와 차이점은 필드(멤버변수)가 final이 아니며, 값을 읽는 getter메서드는 getXxx()와 값을 저장하는 setter메서드가 setXxx()로 자동 생성된다는 것이다.

롬복을 사용할려면 설치 과정이 필요하다. 먼저 https://projectlombok.org/download **에서 최신 버전의 lombok.jar 파일을 다운 받는다.** 다운 받은 경로로 탐색기에서 이동한 후 해당 주소 경로에서 cmd를 입력하고 명령 프롬프트를 실행시킨다.

그리고 다음 명령어를 입력한다. 이 명령어를 실행하기 전 먼저 해당개발툴을 닫는다.

java -jar lombok.jar

그러면 다음과 같은 그림화면이 나온다.



위 화면에서 이클립스나 STS 실행파일을 추가한다.

그리고 해당 프로젝트를 선택한 다음 마우스 오른쪽 버튼을 클릭한다. New – Folder 메뉴를 선택하고 lib폴더를 생성한다. 그리고 다운로드 받은 lombok.jar를 생성한 lib폴더에 붙여넣기 한다.

해당 프로젝트에서 lombok.jar를 사용하기 위해서 lombok.jar 파일을 선택한 다음 마우스 오른쪽 버튼을 클릭해 Build Path – Add to Build Path를 선택해 준다. 그러면 레퍼런스 라이브러리에 추가된 것을 확인할 수 있다. 그러면 해당 프로젝트에서 롬복 라이브러리를 사용할 수 있게 되는 것이다.

```
package net.daum.dto;

import lombok.Data;

@Data //기본 생성자, canEqual(), equals(), hashCode(), toString(), setter(), getter()메서드 자동생성
@AllArgsConstructor //모든 필드(멤버변수)를 초기화시키는 생성자 자동생성
public class MemberDTO {

	private String id;
	private String name;
	private int age;
```

```
}
```

레코드 패턴

자바 14부터 사용할 수 있는 레코드가 있다. 레코드는 불변의 객체이다. 생성자를 통한 레코드가 생성될 때 전달받은 매개값은 필드값으로 세팅되고, 사용 중일때는 필드값을 변경할 수 없다. 레코드는 주로 데이터 전달을 위한 DTO(Data Transfer Object)용으로 사용한다.
자바 21에서 지원하는 레코드 패턴은 레코드의 필드값을 분해해서 변수에 대입하는 기능을 제공한다. 레코드의 필드값을 얻기 위해 Getter를 호출할 필요 없이 매개변수로 바로 받을 수 있어 매우 편리해졌다.

레코드 패턴은 instanceof 연산자와 switch 문의 레이블에서 다음과 같이 작성할 수 있다.

```
 if(obj instanceof 레코드(타입 변수, 타입 변수) {
   변수 사용          레코드 패턴
  }
 switch(obj) {
  case 레코드(타입 변수, 타입 변수) -> {
    //변수 사용
    }
}
```

```
package 새로운문법;

import net.daum.dto.Rectangle;

public class Java21RecordExample {//자바 21의 레코드 패턴으로 변경하면 Getter를 통한 각 필드값을 얻는 코드를 줄일 수 있다.
        private static void area(Object obj) {
                if(obj == null) {
                        return;
                }else if(obj instanceof Rectangle(int width, int height)) {//레코드 패턴 사용
                        System.out.println("사각형 면적 = "+ (width*height));
                }
        }

        //switch문 레코드 패턴으로 변경
        private static void area2(Object obj) {
```

```
                switch(obj) {
                 case Rectangle(int width, int height) -> System.out.println("사각
형 넓이 : "+ (width*height));
                 case null, default -> System.out.println("unKnown");
                }
        }
        public static void main(String[] args) {
         Rectangle rect = new Rectangle(10, 5);
         area(null);
         area(rect);
         System.out.println();
         area2(rect); area2(null);
        }
}
```