



한림대학교 SW중심대학

# 클래스 다이어그램

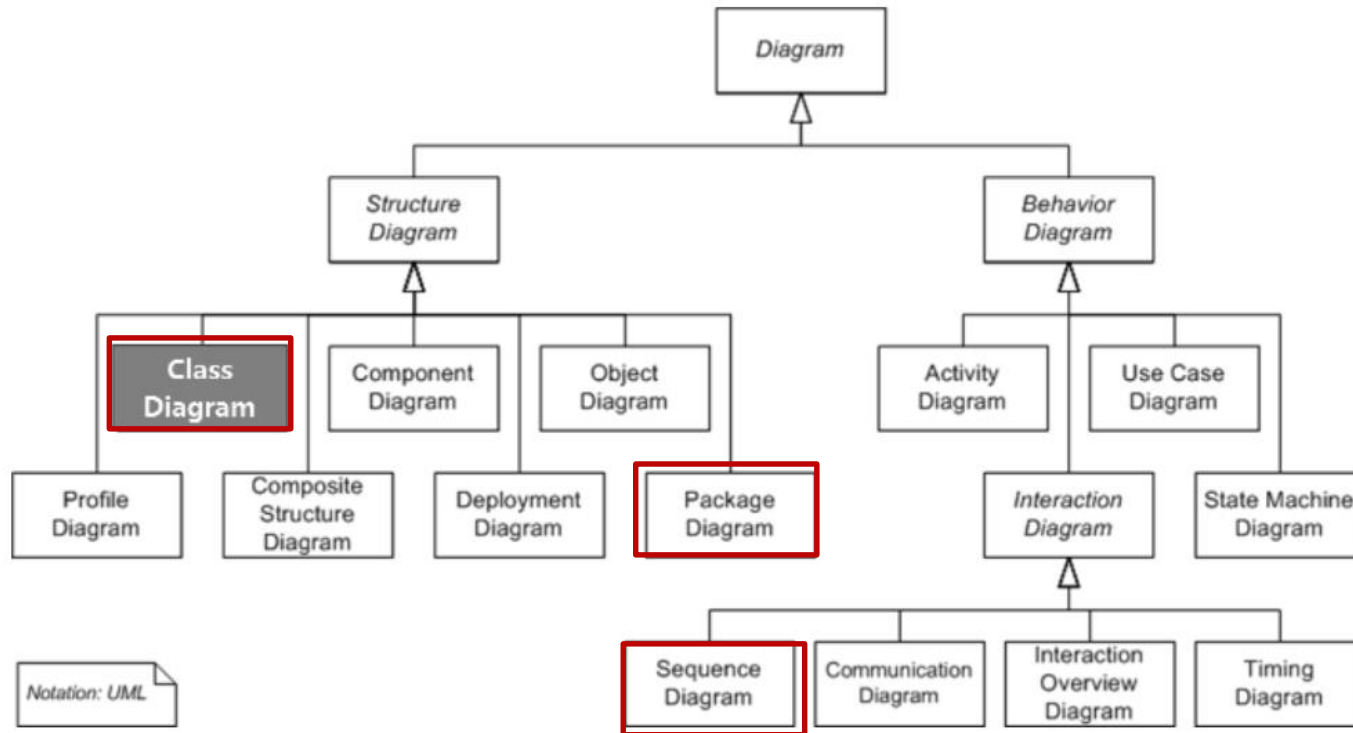
---

# 클래스 다이어그램 - 1

## 생성 및 필드 추가

---

## ■ UML(Unified Modeling Language) 다이어그램의 종류



### ■ structure diagram

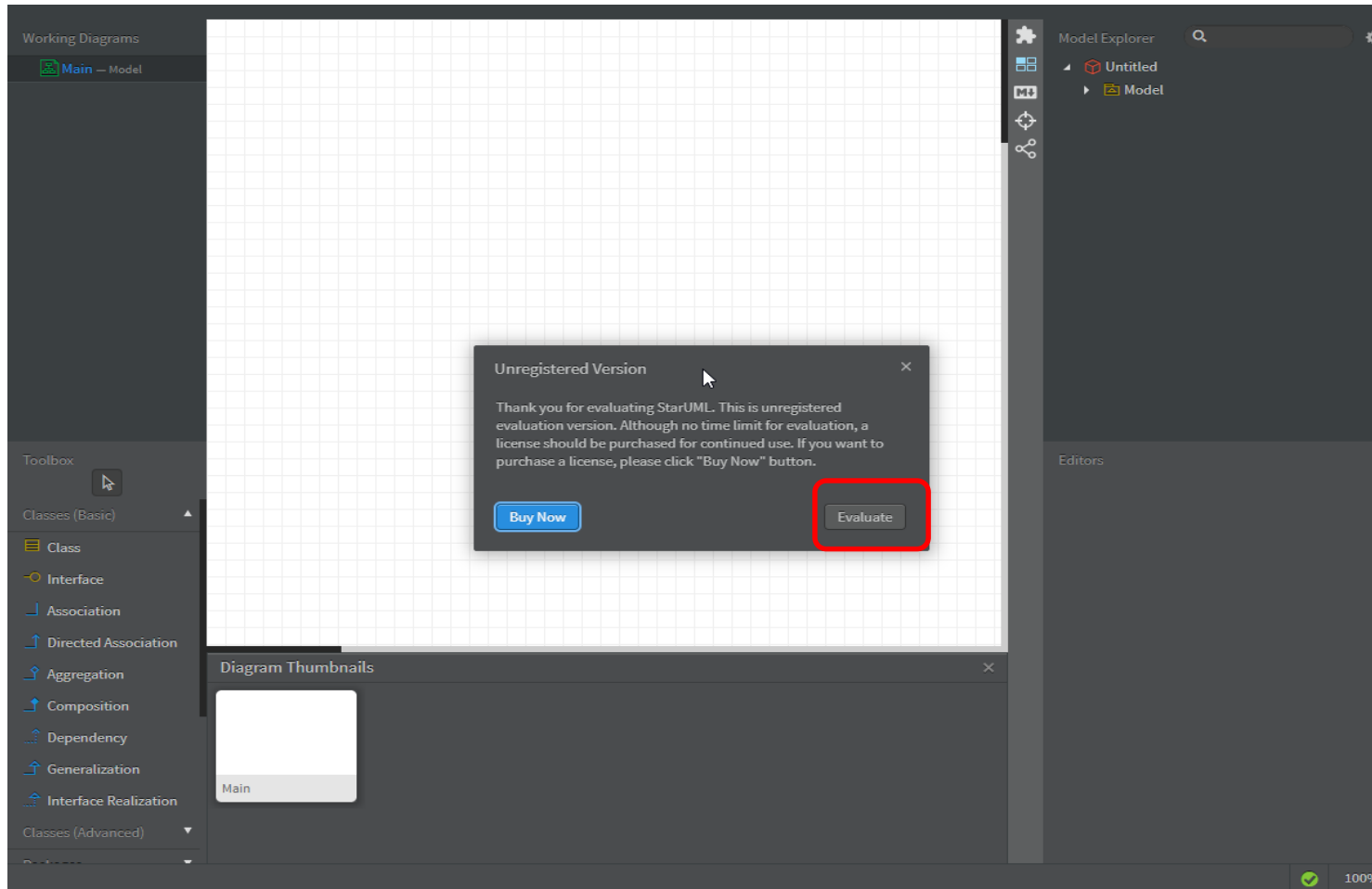
- 시스템의 관계, 개념등의 측면에서 요소들을 나타냄

### ■ behavior diagram

- 각 요소들 혹은 요소들간의 변화나 흐름 주고 받는 데이터 등의 동작을 나타냄

# 클래스 다이어그램

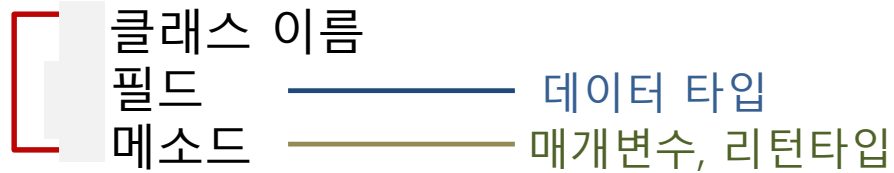
- <https://staruml.io/download>



# 클래스 다이어그램

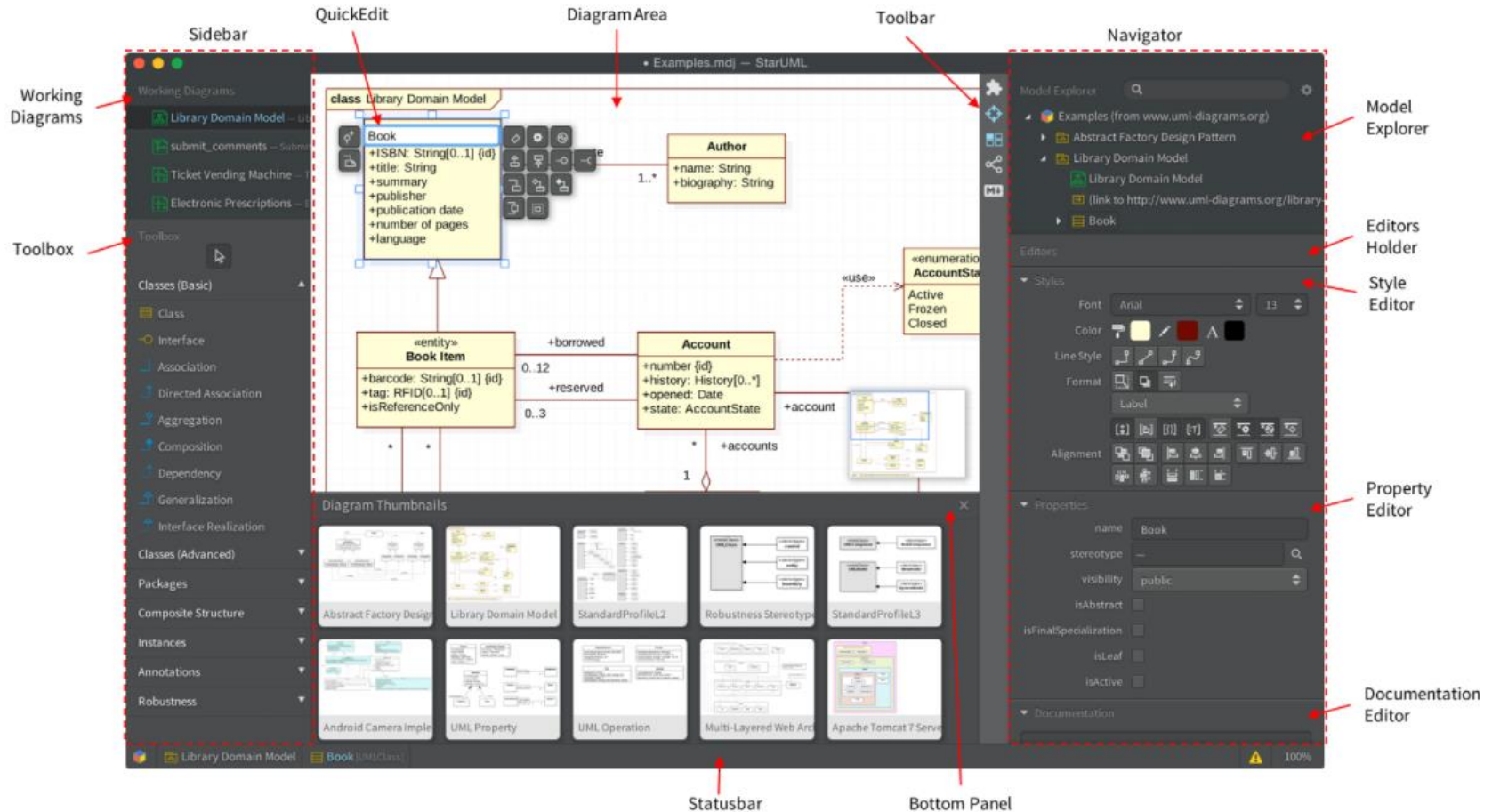
- 다이어그램으로 클래스 내부 구성요소 및 클래스 간의 관계를 표기하는 다이어그램
- 시스템의 일부 또는 전체의 구조를 나타낼 수 있음
  - 의존관계를 알 수 있어 순환 의존 등의 좋지 않은 구조를 알 수 있음
- 클래스 다이어그램의 요소

접근 제한자



# 클래스 다이어그램

## ■ 창 설명



# 클래스 다이어그램

StarUML (UNREGISTERED)

File Edit Format Model Tools View Window Debug Help

Working Diagrams

Main — Model

2. 에디터 창에 클릭

1. 필요한 요소 클릭

Model Explorer

Untitled

Model

Main

Class1

Toolbox

Classes (Basic)

Class

Interface

Association

Directed Association

Aggregation

Composition

Dependency

Generalization

Interface Realization

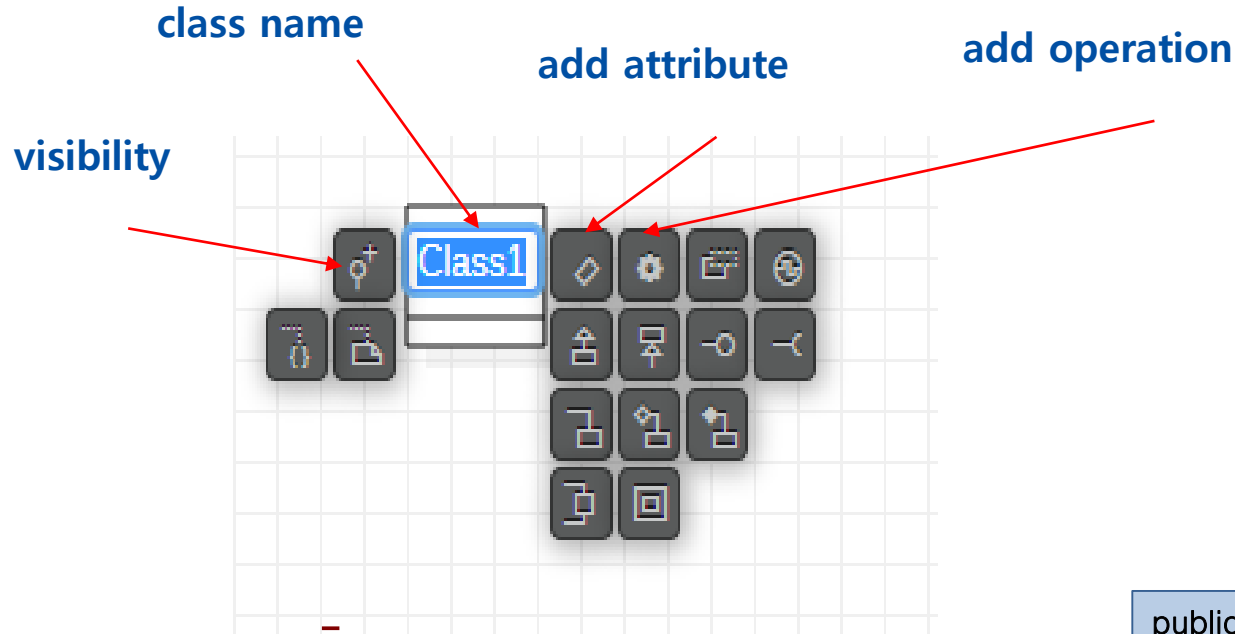
Classes (Advanced)

Editors

중요 : 모든 요소의 삭제는 model explore에서 diagram area에서 하지 않음

# 클래스 다이어그램

## ■ 클래스의 구성 요소의 표현



클래스이름	User
+멤버변수1 +멤버변수2	-pw: String +id: String
+메소드1() +메소드2()	+getPw(): String +setPw(pw: String): void

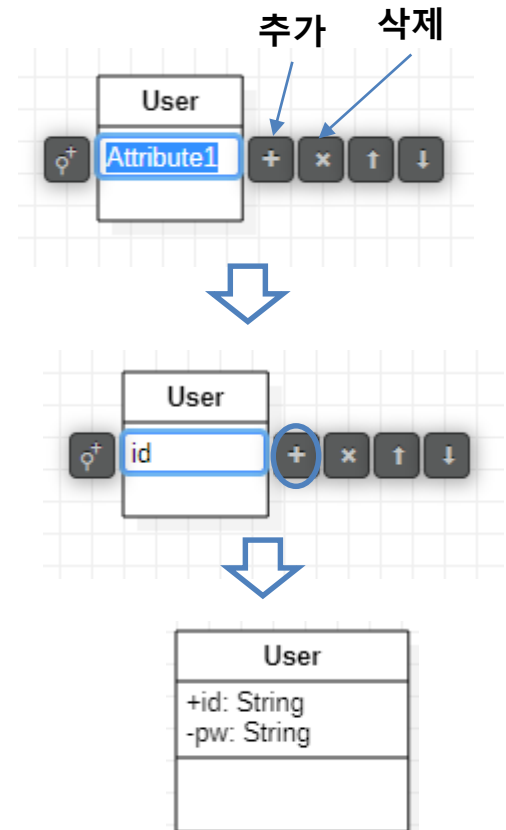
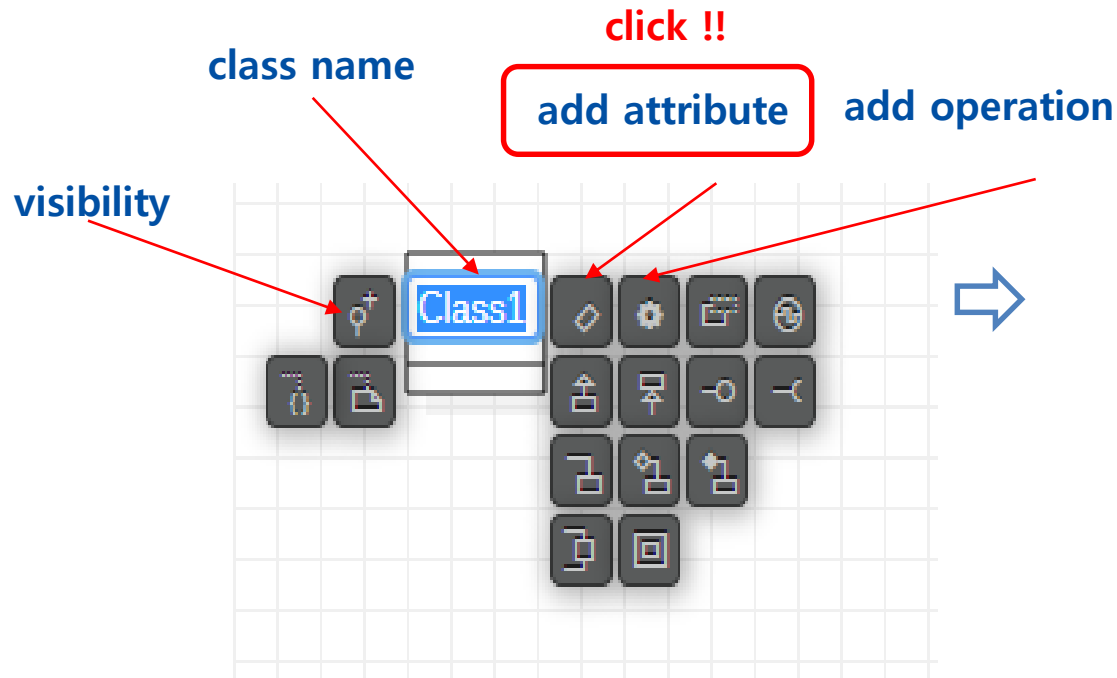
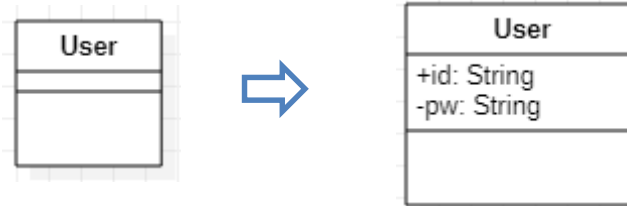
```
public class User {  
  
    public String id;  
    private String pw;  
  
    public String getPw() {  
        return pw;  
    }  
    public void setPw(String pw) {  
        this.pw = pw;  
    }  
  
}
```



# 클래스 다이어그램

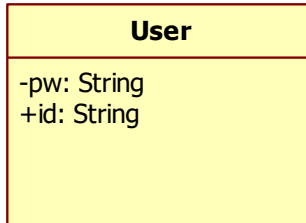
## ■ 멤버변수 추가

- 형식 : 변수명 : 데이터타입



# 클래스 다이어그램

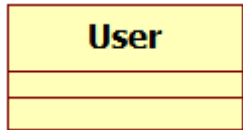
-



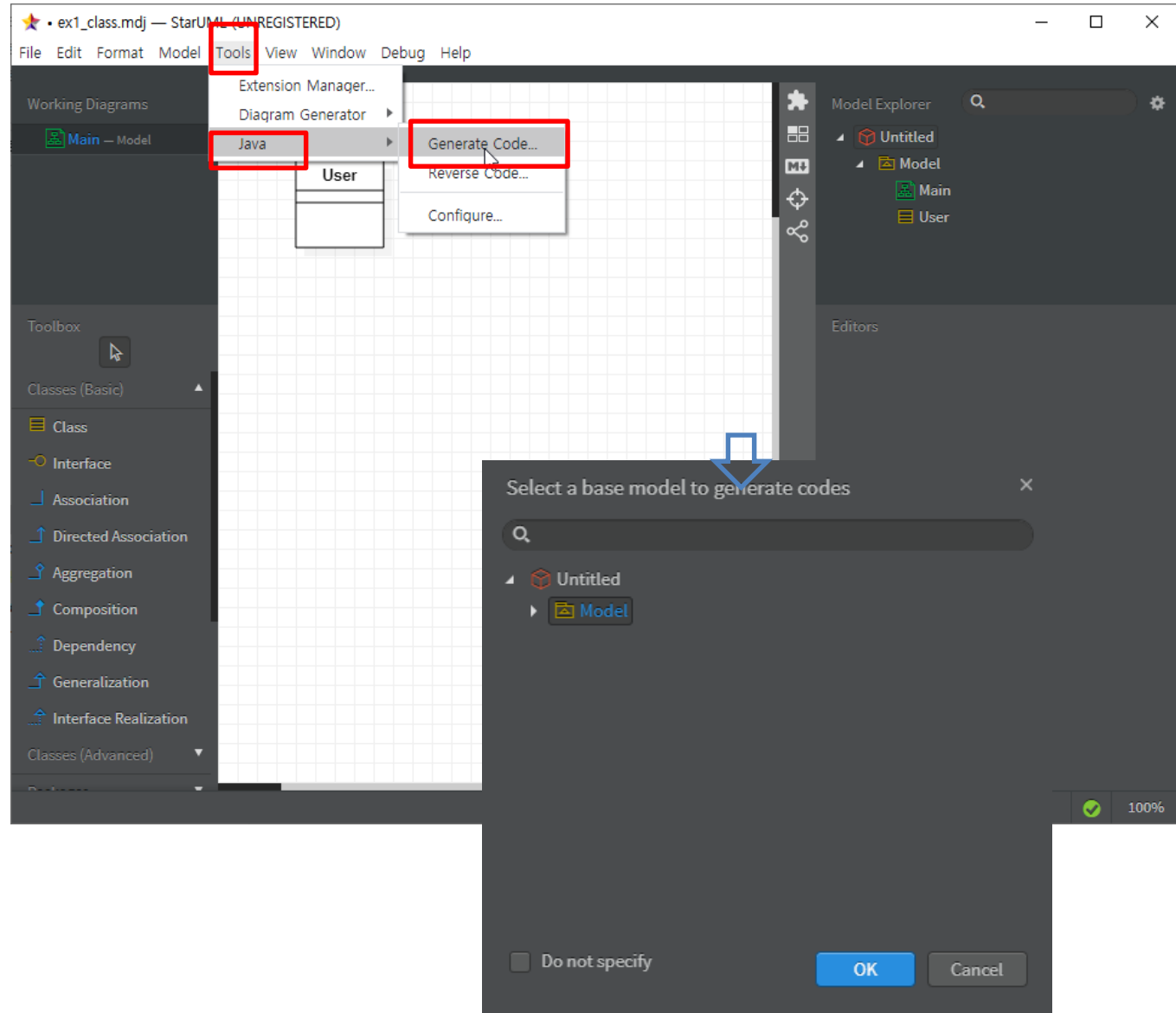
```
public class User {  
  
    public String id;  
    private String pw;  
  
}
```

# 클래스 다이어그램

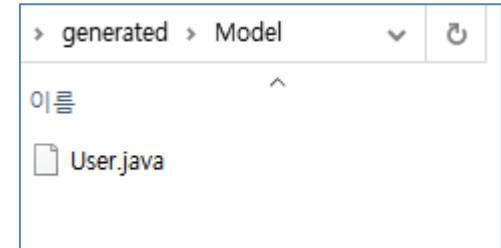
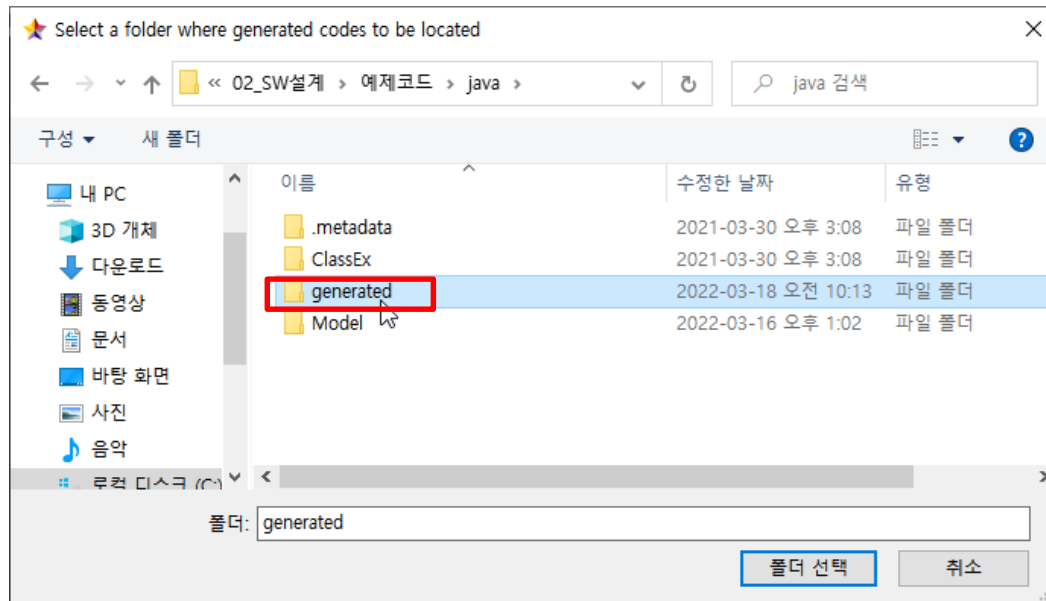
## ■ 클래스 생성 및 자바 코드 생성



```
public class User {  
  
}
```



# 클래스 다이어그램



```
import java.util.*;

/**
 *
 */
public class User {

    /**
     * Default constructor
     */
    public User() {
    }

}
```



한림대학교 SW중심대학

# 클래스 다이어그램 - 메소드

---

# 클래스 다이어그램

## ■ 메소드

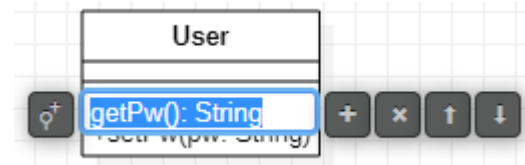
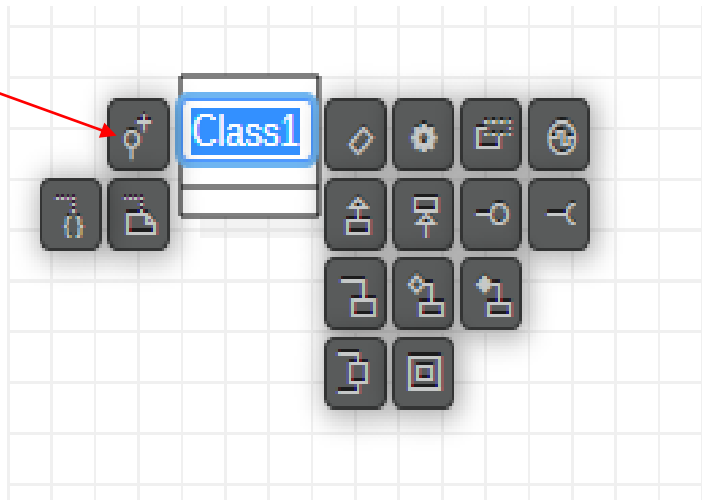
- 형식 : **메소드명(매개변수명 : 데이터타입):리턴타입**

메소드 형식

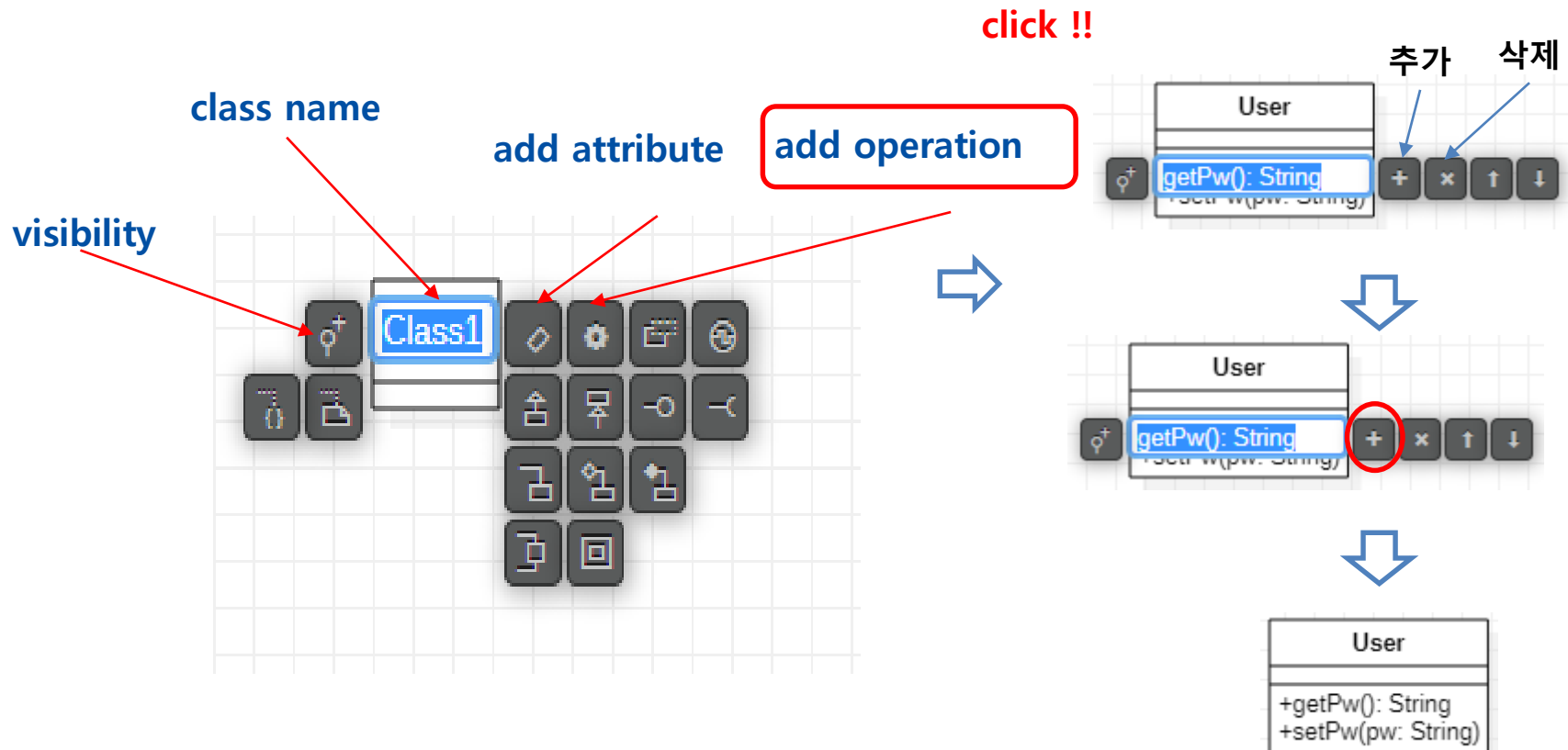
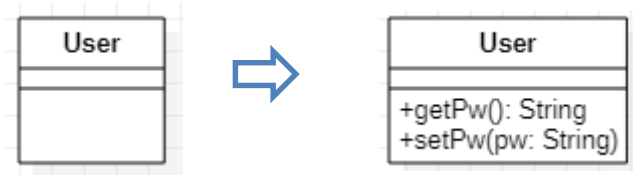
접근제한자 리턴타입 메소드명(매개변수들){  
}

```
public class User {  
    public String getPw() {  
  
    }  
  
    public void setPw(String pw) {  
  
    }  
}
```

visibility

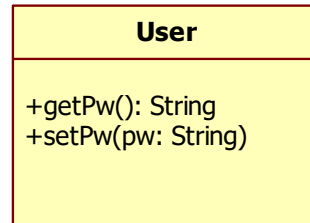


# 클래스 다이어그램



# 클래스 다이어그램

-



```
public class User {  
    public String getPw() {  
    }  
  
    public void setPw(String pw) {  
    }  
}
```



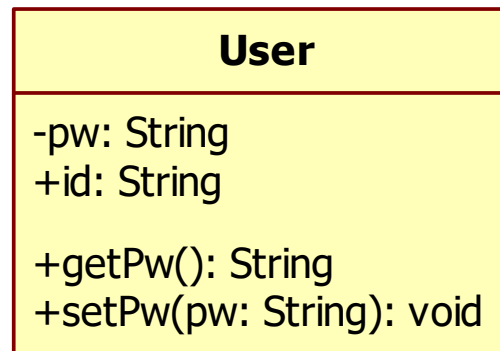
# 클래스 다이어그램

```
public class User {  
  
    public String id;  
    private String pw;  
  
    public String getPw() {  
        return pw;  
    }  
    public void c(String pw) {  
        this.pw = pw;  
    }  
  
}
```

내가 만들고 싶은 코드의 형태

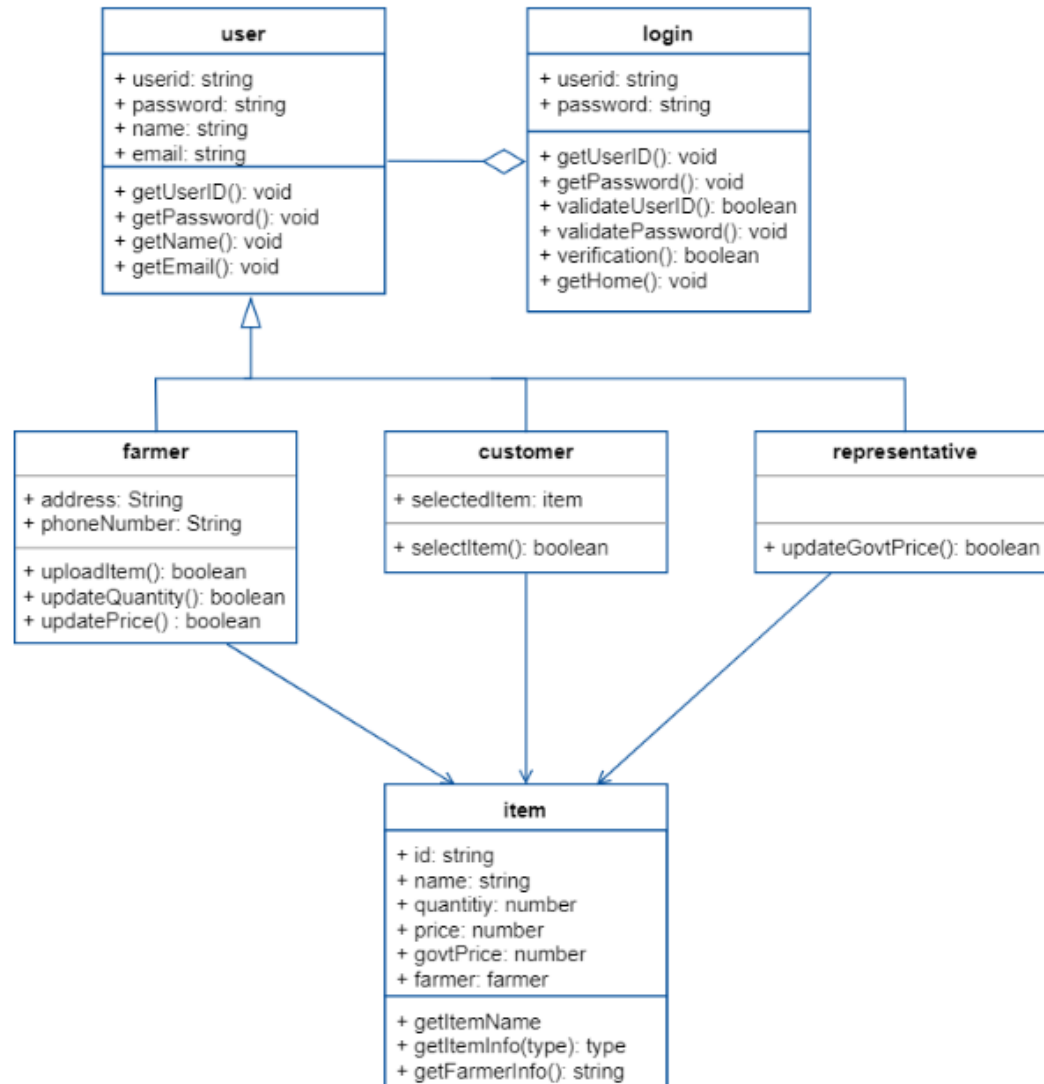
```
public class User {  
  
    private String pw;  
    public String id;  
  
    public String getPw() {  
          
    }  
    public void setPw(String pw) {  
          
    }  
  
}
```

클래스 다이어그램으로 생성된 코드



Class diagram

# 클래스 사이의 관계











한림대학교 SW중심대학

# 클래스 다이어그램 Generalization

---

# 클래스 사이의 관계

Class Diagram Relationship Type	Notation
Association 연관	
Generalization 상속	
Realization/ Implementation 실체화	
Dependency 의존성	
Aggregation 집합, 집합연관	
Composition 합성, 복합연관	

# Generalization

## ■ 일반화(Generalization) 관계

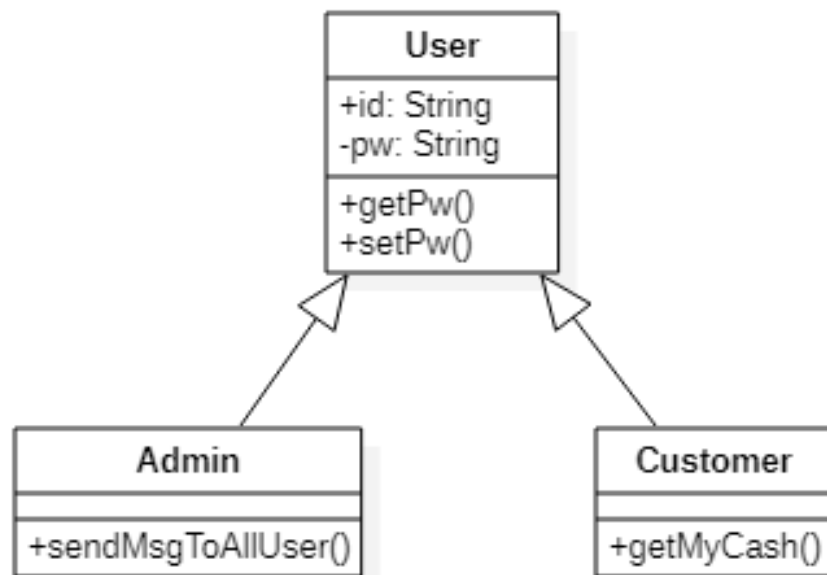
- 슈퍼클래스와 서브 클래스간의 상속관계를 나타냄

```
public class User {  
  
    private String id;  
    private String pw;  
  
    public String getPw() {  
        return pw;  
    }  
    public void setPw(String pw) {  
        this.pw = pw;  
    }  
  
}
```

```
public class Admin extends User{  
  
    public boolean sendMsgToAllUser (String msg) {  
  
    }  
  
}
```

```
public class Customer extends User{  
  
    public double getMyCash() {  
  
    }  
  
}
```

# Generalization



- └ Association
- └ DirectedAssociation
- └ Aggregation
- └ Composition
- └ **Generalization**
- └ Dependency
- └ Realization

# Generalization

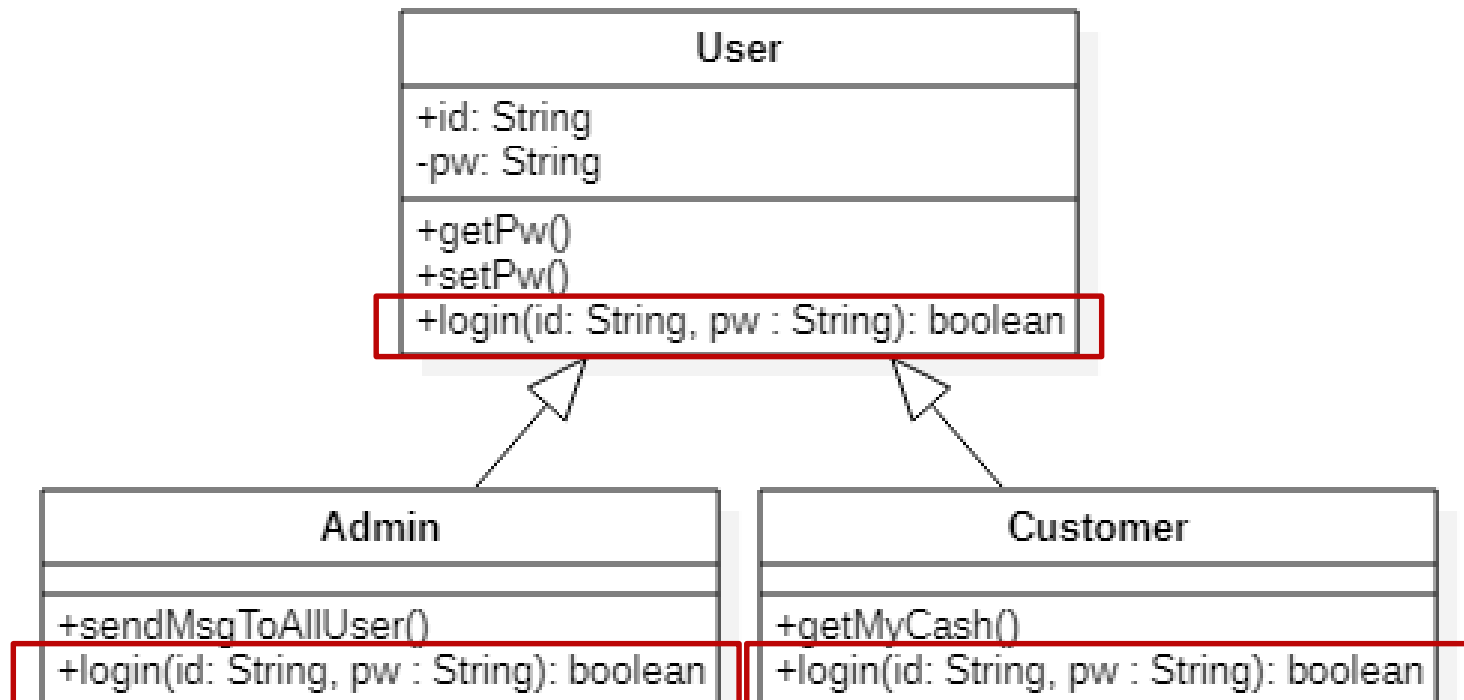
## ■ method overriding

```
public class User {  
  
    private String id;  
    private String pw;  
  
    public String getPw() {  
        return pw;  
    }  
    public void setPw(String pw) {  
        this.pw = pw;  
    }  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        //login 구현  
        return result;  
    }  
}
```

```
public class Admin extends User{  
  
    public boolean sendMsgToAllUser (String msg) {  
  
    }  
  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        //Admin login 구현  
        return result;  
    }  
}
```

```
public class Customer extends User{  
  
    public double getMyCash() {  
  
    }  
  
}
```

# Generalization





# 클래스 다이어그램 Realization

---

# Realization

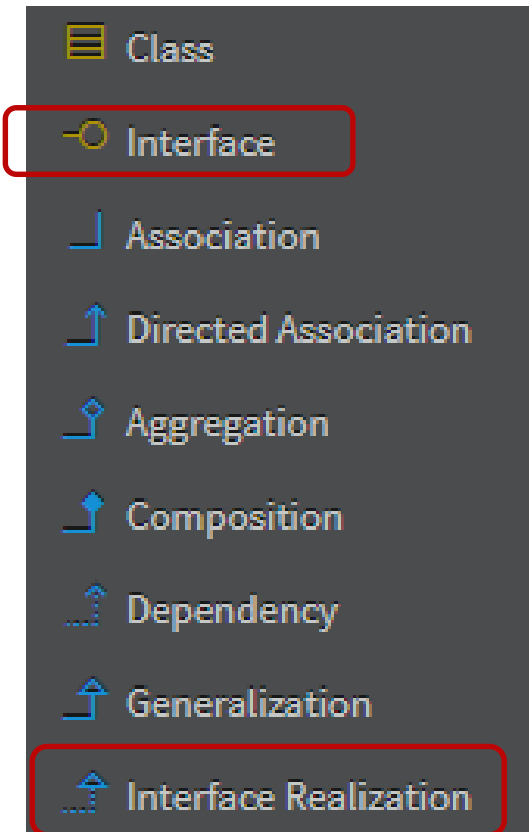
- 인터페이스와 그 구현 클래스와의 관계를 나타냄

```
public class Calculator {  
    public String calName;  
  
    public int add(int x, int y) {  
        return x+y;  
    }  
}
```

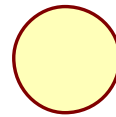
```
interface Calculator {  
    int DecimalUnit = 4;  
    int add(int x, int y) ;  
}
```

```
public class CalculatorImpl implements Calculator {  
  
    public int add(int x, int y) {  
        return x+y;  
    }  
}
```

# Realization



1



**CheckeckLogic**  
interface icon

2

**<<interface>>  
CheckeckLogic**  
interface strereo

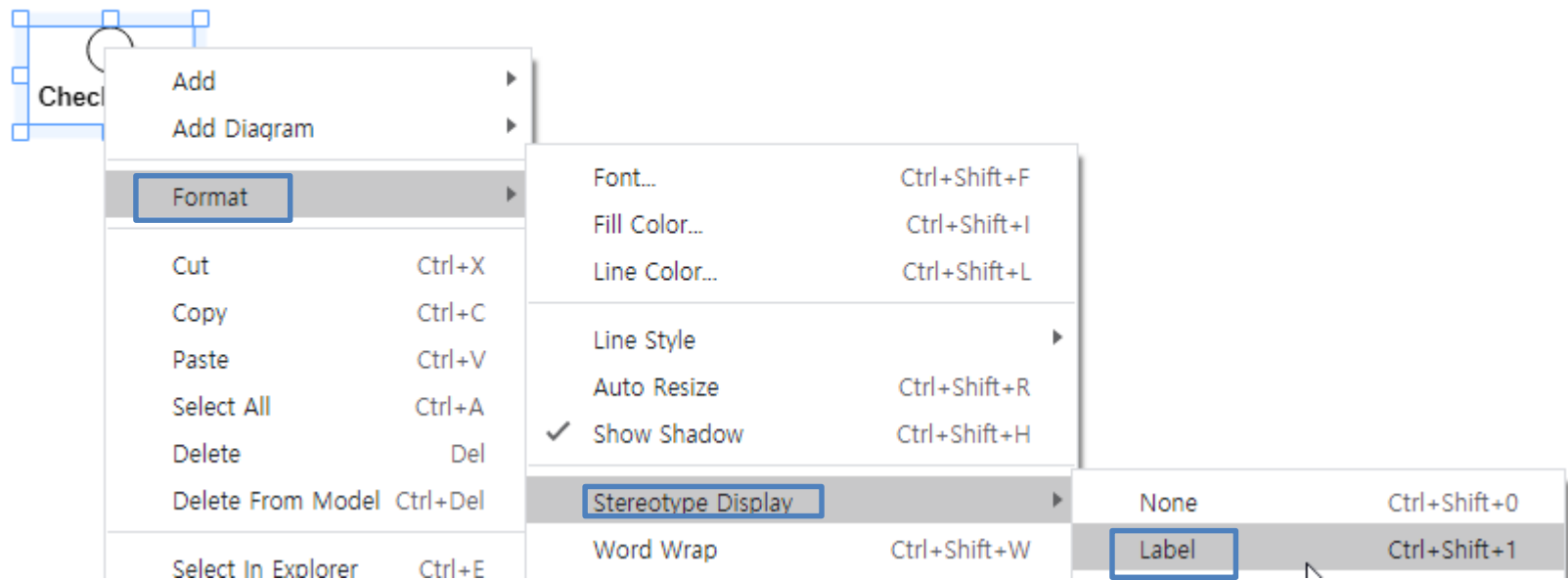
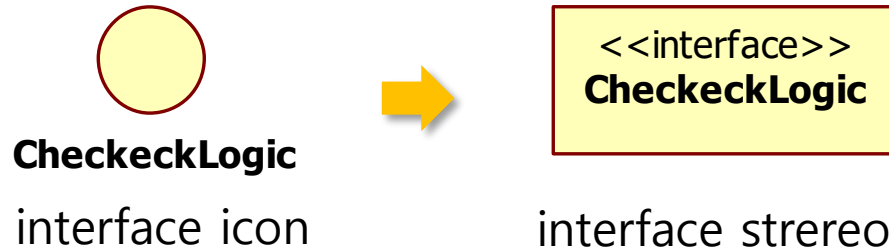
# stereo type

- UML에서 제공하는 기본 요소(표현의 한계를 가짐) 이외에 추가적인 확장 요소를 나타내는것
- <<>> 로 표기

표현	뜻
«interface»	인터페이스 클래스
«abstract»	추상화 클래스
«enumeration»	열거형 타입 클래스
«utility»	인스턴스가 없는 static 메서드만 모아둔 클래스
«create»	생성자

# Realization

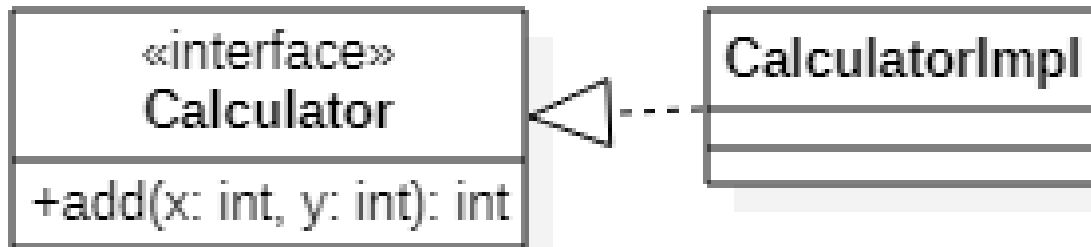
## ■ 아이콘 타입을 스테레오 타입으로 바꾸는 방법



# Realization

```
interface Calculator {  
    int add(int x, int y) ;  
}
```

```
public class CalculatorImpl implements Calculator {  
  
    public int add(int x, int y) {  
        return x+y;  
    }  
}
```





한림대학교 SW중심대학

# 클래스 다이어그램 dependency, association

---

## ■ dependency

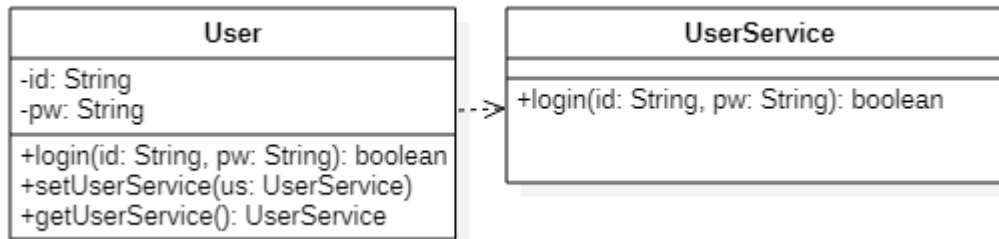
- 한 클래스의 메소드에서 다른 클래스를 사용하는 경우
  - 매개변수로 사용될 때
  - 메서드 내부의 지역객체로 참조될 때
  - 리턴 값으로 명시되어 사용될 경우

```
public class User {  
  
    private String id;  
    private String pw;  
  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        UserService us = new UserService();  
        result=us.login(id, pw);  
        return result;  
    }  
  
    public void setUserService(UserService us) {  
  
    }  
    public UserService getUserService(){  
        return new UserService();  
    }  
}
```

```
public class UserService {  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        //ToDo : implementation  
        return result;  
    }  
}
```



# dependency



- └ Association
- └ DirectedAssociation
- └ Aggregation
- └ Composition
- └ Generalization
- └ **Dependency**
- └ Realization



한림대학교 SW중심대학

# 클래스 다이어그램 association

---

# Association

## ■ Association

- 한 객체가 다른 객체와 연결되어 있음을 나타낼때
  - 속성으로 다른 다른 클래스 변수를 가지고 있을때
  - 참조의 관계가 모호함

```
public class User {  
    private String id;  
    private String pw;  
  
    public UserService us;  
    public User(){  
        us = new UserService();  
    }  
  
    public boolean login(String id, String pw) {  
        boolean result=false;  
  
        result=us.login(id, pw);  
        return result;  
    }  
}
```

```
public class UserService {  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        //ToDo : implementation  
        return result;  
    }  
}
```

# Association

## Association

```
public class User {  
  
    private String id;  
    private String pw;  
  
    public UserService us;  
    public User(){  
        us = new UserService();  
    }  
  
    public boolean login(String id, String pw) {  
        boolean result=false;  
  
        result=us.login(id, pw);  
        return result;  
    }  
}
```

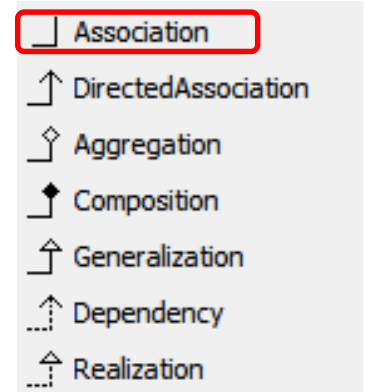
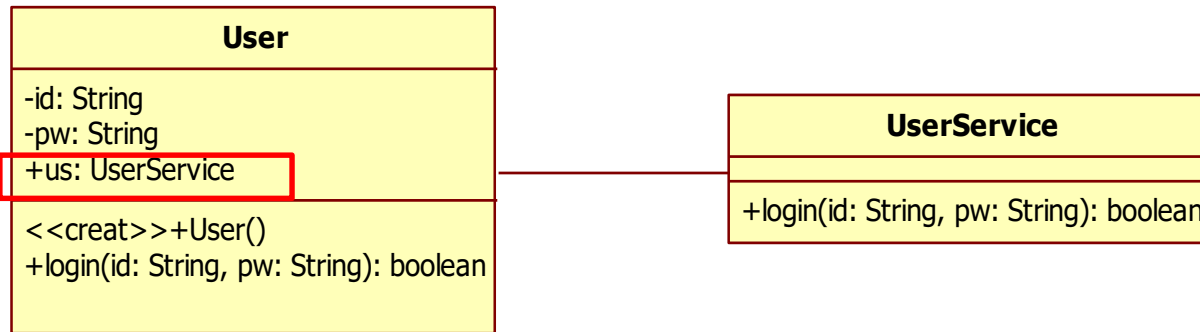
- 다른 클래스 객체를 필드로 가지고 있을때

## Dependency

```
public class User {  
  
    private String id;  
    private String pw;  
  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        UserService us = new UserService();  
        result=us.login(id, pw);  
        return result;  
    }  
  
    public void setUserService(UserService us) {  
  
    }  
}
```

- 매개변수 (참조값)로 사용될 때
- 메서드 내부의 지역객체로 참조될 때

# Association



# DirectedAssociation

## ■ DirectedAssociation

- Association의 의미에 참조하는쪽과 당하는 쪽의 방향성이 추가됨

```
public class User {  
  
    private String id;  
    private String pw;  
  
    public UserService us;  
    public User(){  
        us = new UserService();  
    }  
  
    public boolean login(String id, String pw) {  
        boolean result=false;  
  
        result=us.login(id, pw);  
        return result;  
    }  
}
```

```
public class UserService {  
    public boolean login(String id, String pw) {  
        boolean result=false;  
        //ToDo : implementation  
        return result;  
    }  
}
```

