SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

Java

SAMSUNG SW ACADEMY FOR YOUTH

Programming Language

객체지향 프로그래밍

- UML 클래스 다이어그램
- 상속
- Object 클래스
- final 키워드

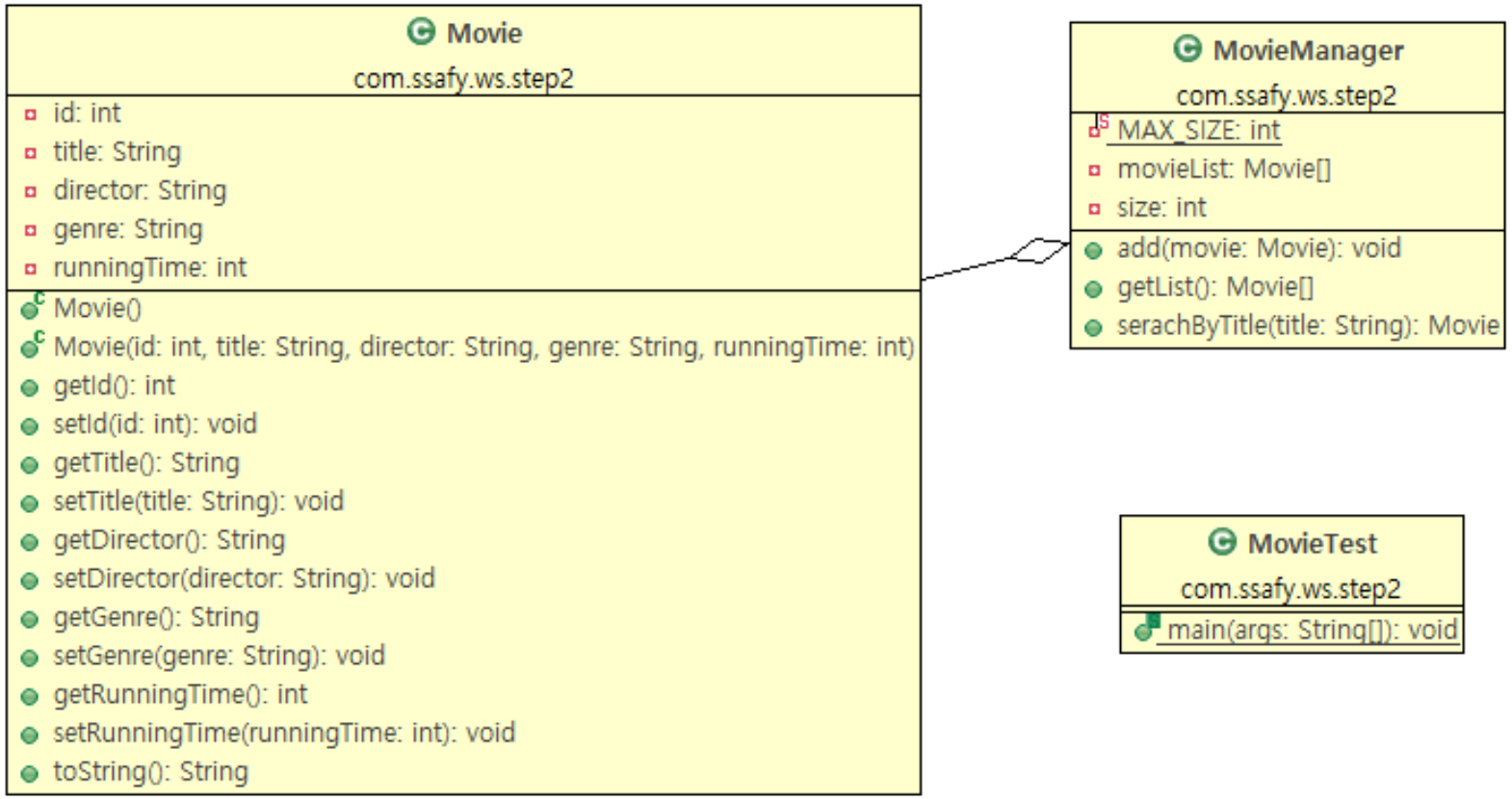
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

UML 클래스 다이어그램

UML 클래스 다이어그램

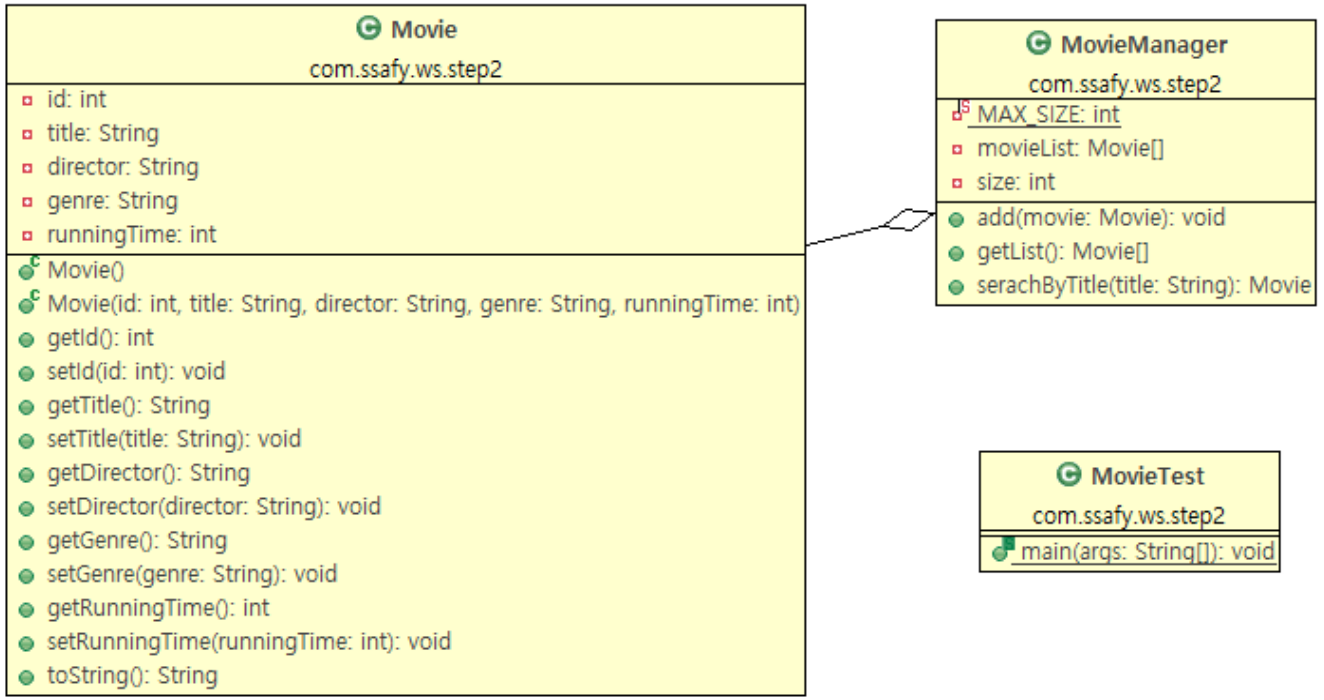
- 다음 UML 클래스 다이어그램을 해석해 볼까요?



UML 클래스 다이어그램

Confidential

- 다음 UML 클래스 다이어그램을 해석해 볼까요?



	Class (public)
	Interface (public)

C	Constructor
A	Abstract member
F	Final member
S	Static member
D	Default method

	Default field (package visible)
	Private field
	Protected field
	Public field
	Default method (package visible)
	Private method
	Protected method
	Public method

https://help.eclipse.org/latest/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/ref-icons.htm

UML 클래스 다이어그램

UML 클래스 다이어그램

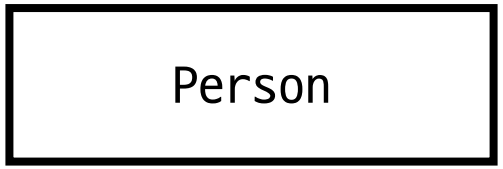
- 통합 모델링 언어(UML : Unified Modeling Language, ISO/IEC 19501 표준)에서 정의한 다이어그램
- 많은 종류의 다이어그램이 정의되어 있고, 클래스 다이어그램은 그 중 하나

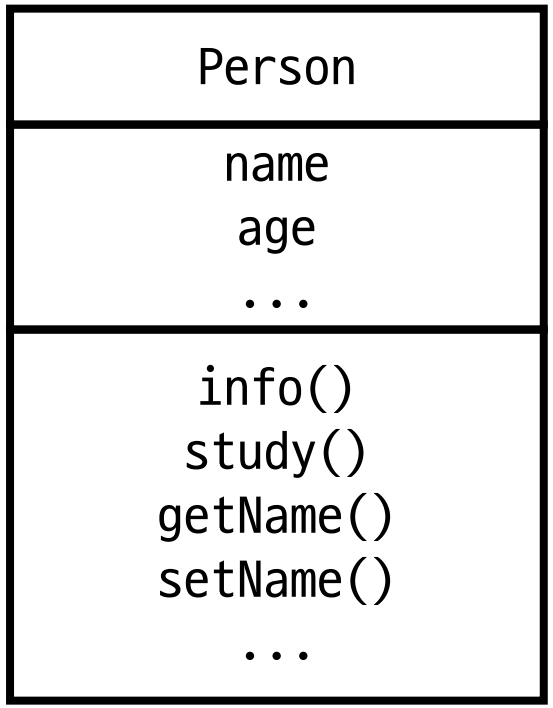
패키지 & 클래스

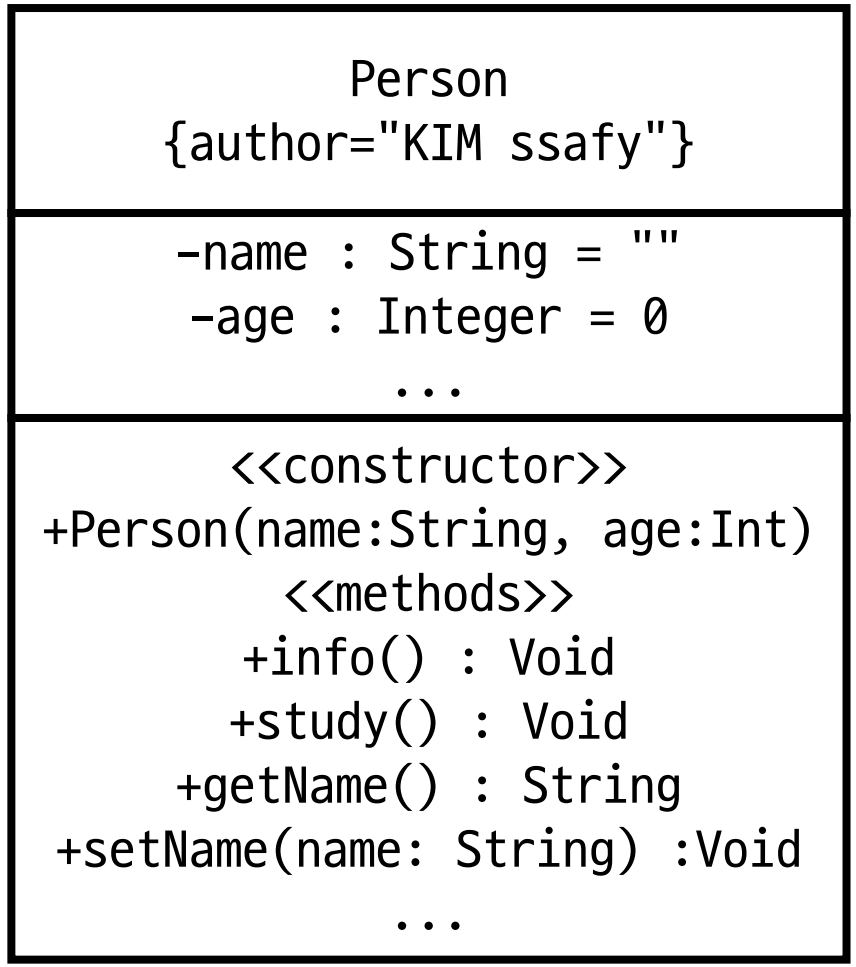
UML 클래스 다이어그램

클래스

- 사각형 모양으로 나타냄.
- Level of Detail 에 따라 다양한 모습으로 나타낼 수 있음.
- visibility는 다음 중 하나
  + public visibility
  # protected visibility
  - private visibility
  ~ package visibility

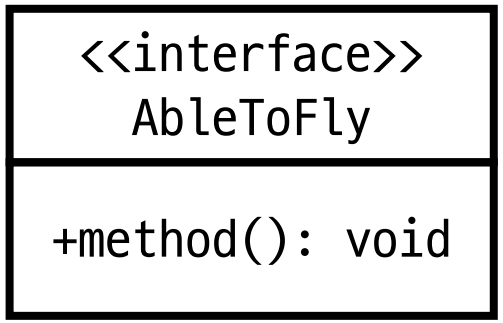


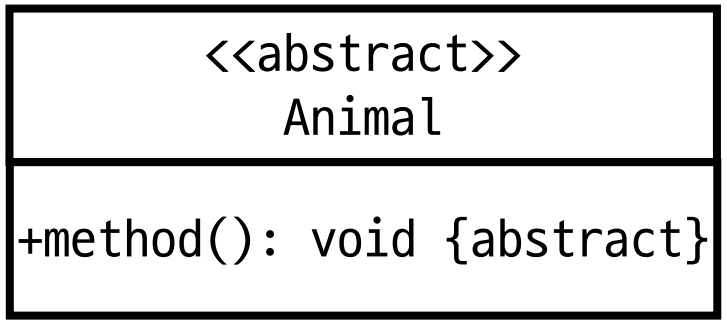


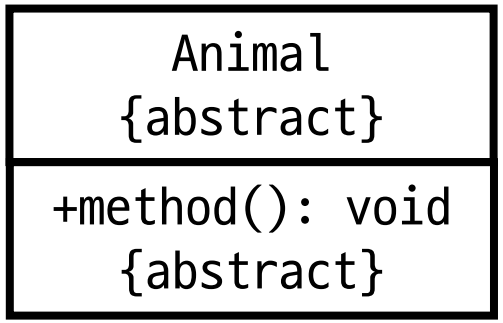


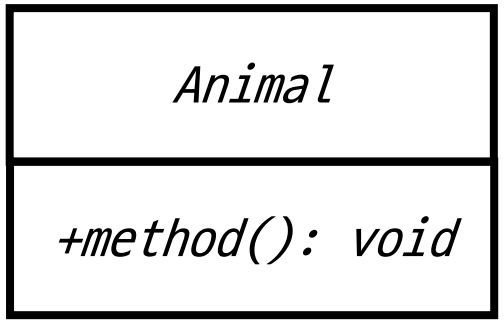
UML 클래스 다이어그램

- 인터페이스, 추상 클래스
  - 이름 앞에 인터페이스는 <<interface>>를 추가
  - 추상클래스는 <<abstract>>를 추가하거나
    중괄호 표기법으로 abstract 임을 뒤 또는 아래에 표시하거나
    이텔릭체로 나타냄







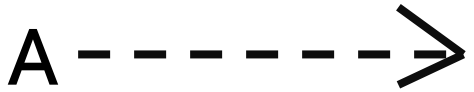


UML 클래스 다이어그램

- 관계: 클래스, 인터페이스 간의 관계를 나타냄

A는 B를 상속(Inheritance)받는다.(A: 자식 클래스, B: 부모 클래스)

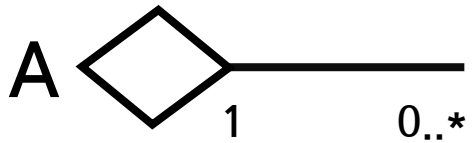
A는 B를 구현(Implementation)한다.(A: 구현클래스, B: 인터페이스)



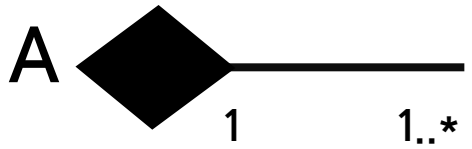
A는 B에 의존(Dependency)한다.

A는 B와 연관(Association)되어 있다.

양방향 연관: A와 B는 서로 연관되어 있다.



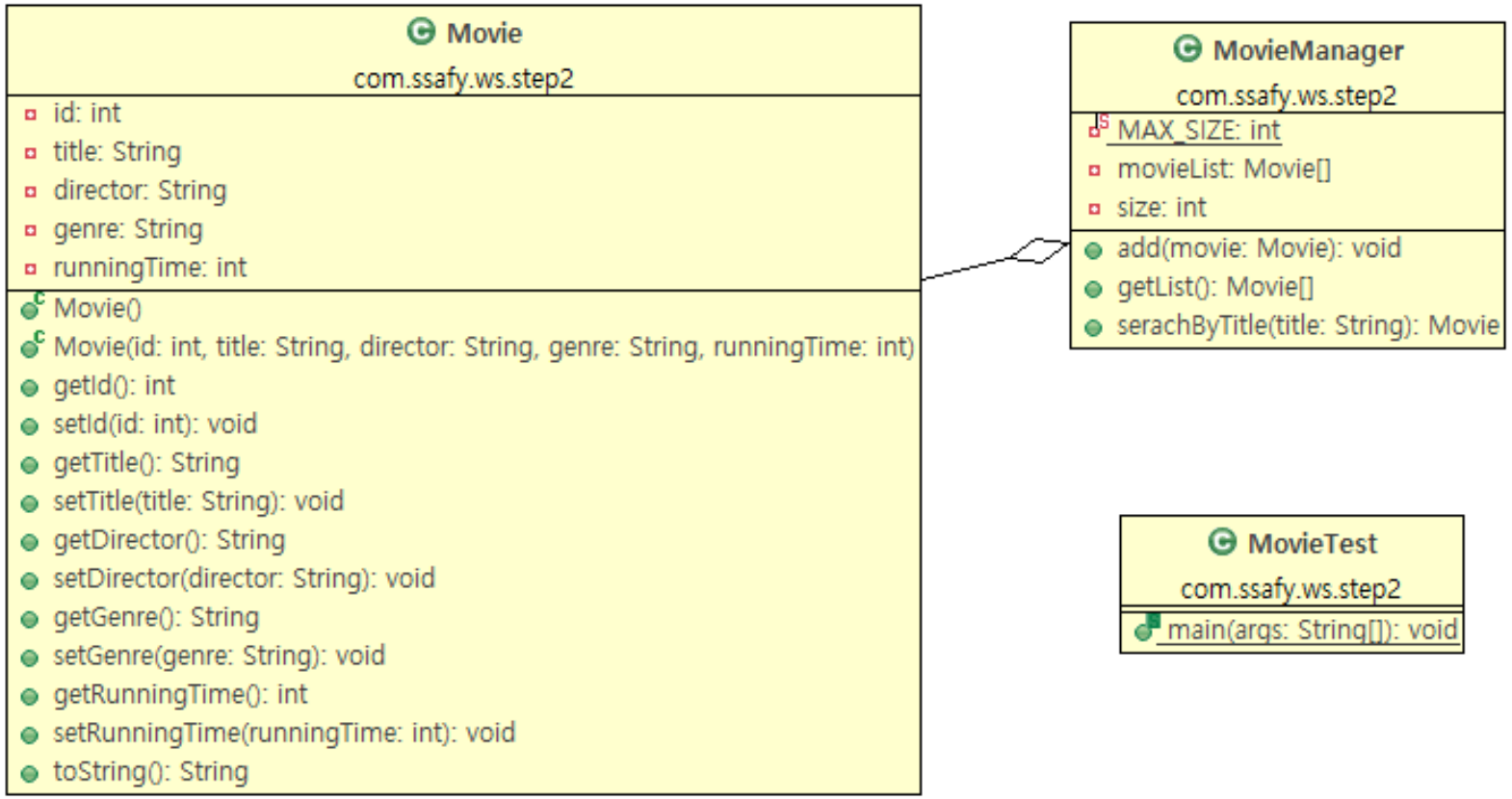
집합(Aggregation): A는 0개 이상의 B를 가질 수 있다.
B가 없어도 A는 존재 가능 ex) StudentManager - Student의 배열



구성(Composition): A는 1개 이상의 B로 구성된다.
B가 없으면 A는 존재 불가: ex) 집 - 방

UML 클래스 다이어그램

- 다음 UML 클래스 다이어그램을 해석해 볼까요?



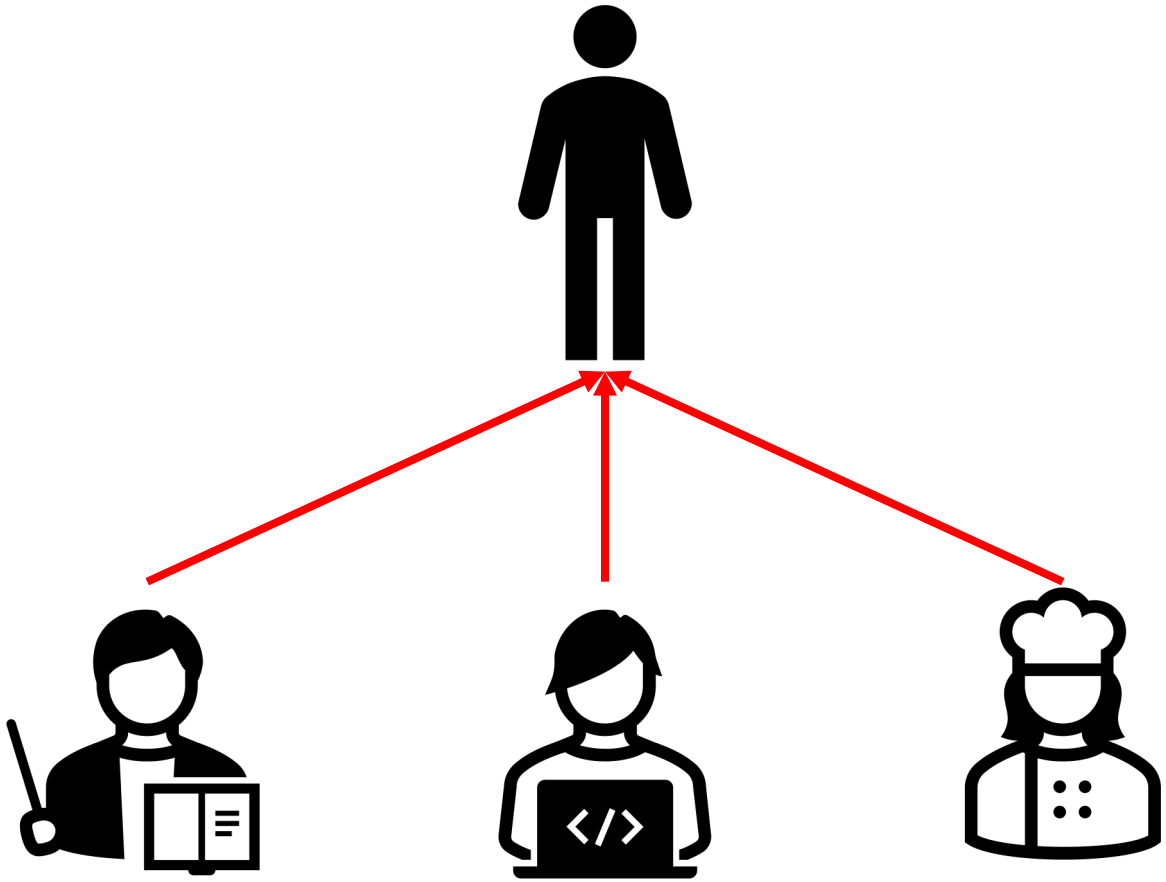
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

상속

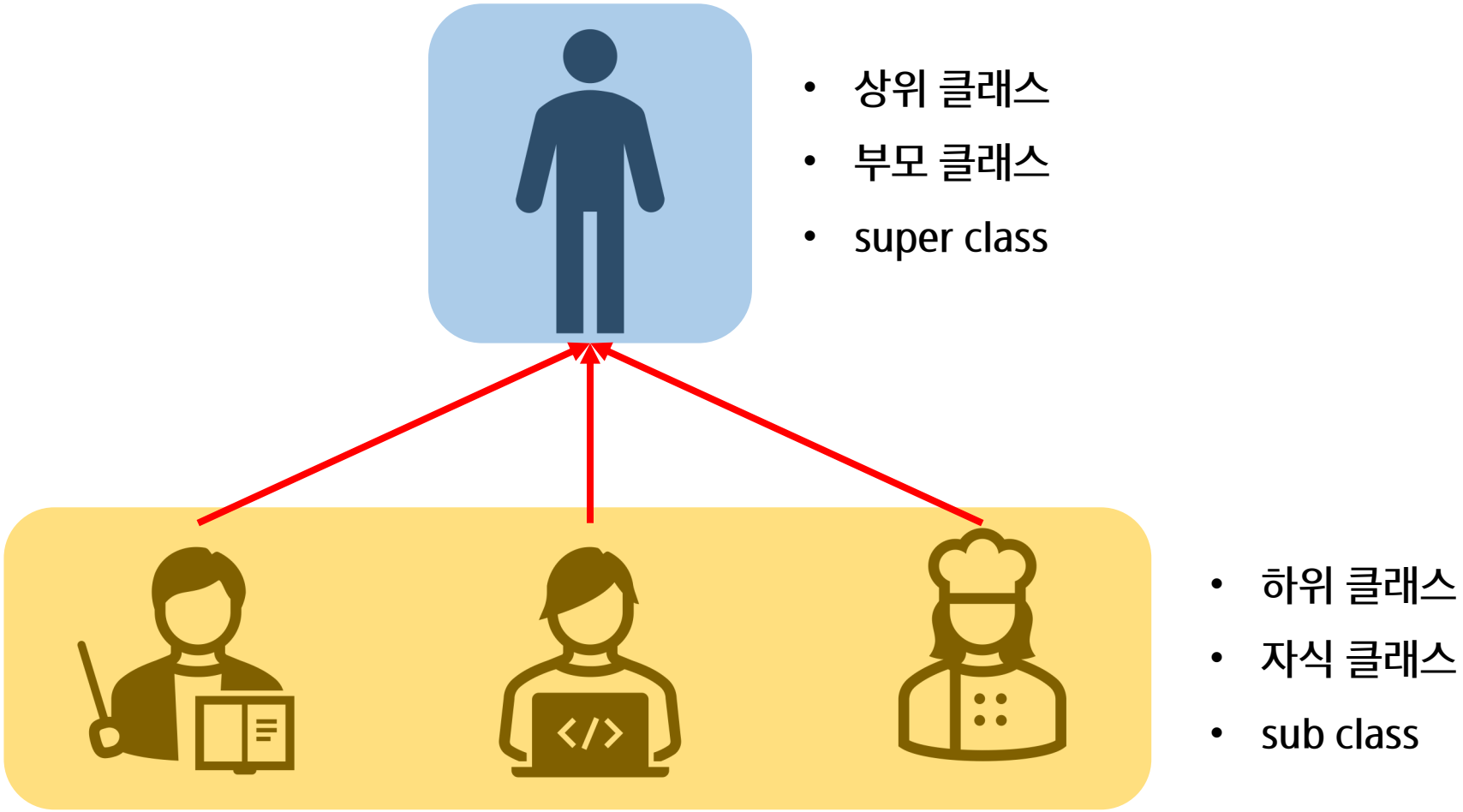
상속(Inheritance)

- 생각해 봅시다.



상속(Inheritance)

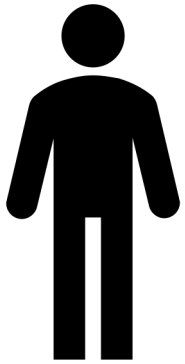
- 상위 클래스의 속성과 메서드를 물려받아 확장하여 새로운 자식 클래스를 정의하는 것

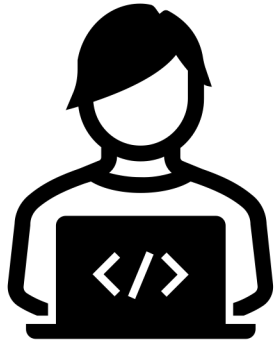


- 상위 클래스
- 부모 클래스
- super class

- 하위 클래스
- 자식 클래스
- sub class

상속(Inheritance)

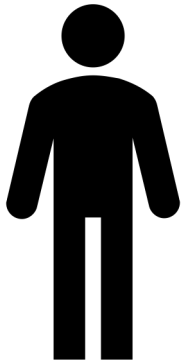


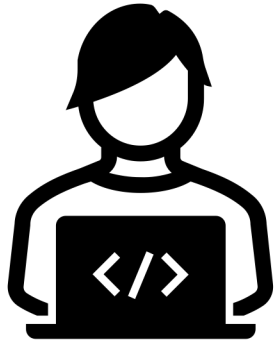


```java
public class Person {
    String name;
    int age;

    public void eat() {
        System.out.println("음식을 먹는다.");
    }
}
```

```java
public class Student {
    String name;
    int age;
    String major;

    public void eat() {
        System.out.println("음식을 먹는다.");
    }

    public void study() {
        System.out.println("공부를 한다.");
    }
}
```

상속(Inheritance)

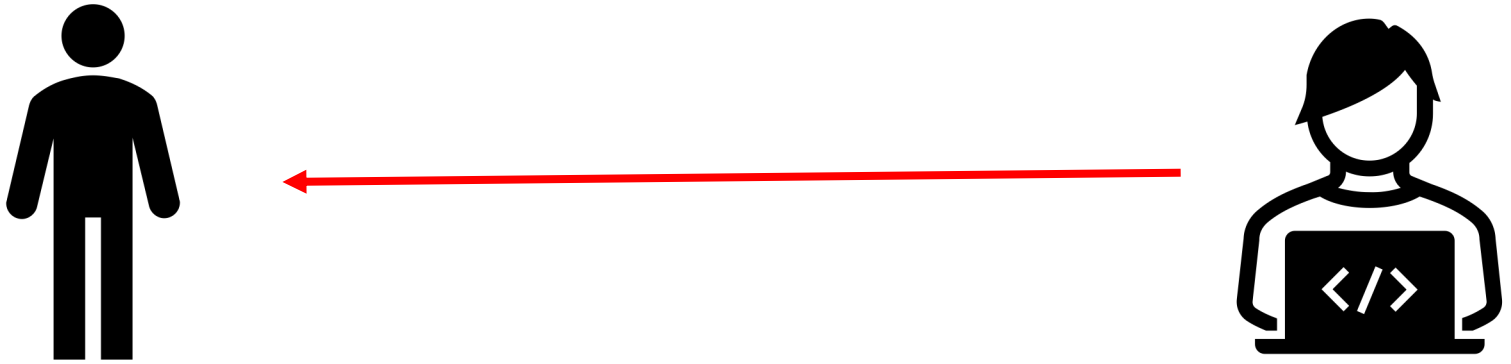




```java
public class Person {
    String name;
    int age;

    public void eat() {
        System.out.println("음식을 먹는다.");
    }
}
```

```java
public class Student {
    String name;
    int age;
    String major;

    public void eat() {
        System.out.println("음식을 먹는다.");
    }

    public void study() {
        System.out.println("공부를 한다.");
    }
}
```

상속(Inheritance)



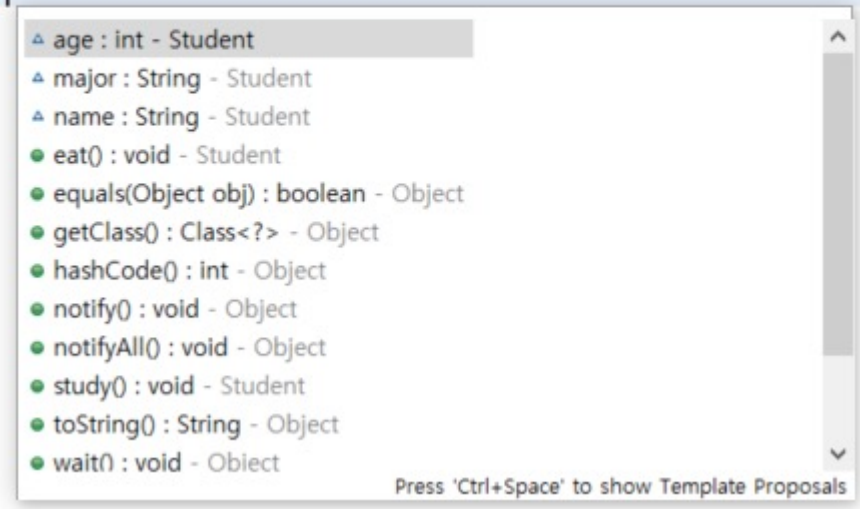
```java
public class Person {
    String name;
    int age;

    public void eat() {
        System.out.println("음식을 먹는다.");
    }
}
```

```java
public class Student extends Person {
    String major;

    public void study() {
        System.out.println("공부를 한다.");
    }
}
```

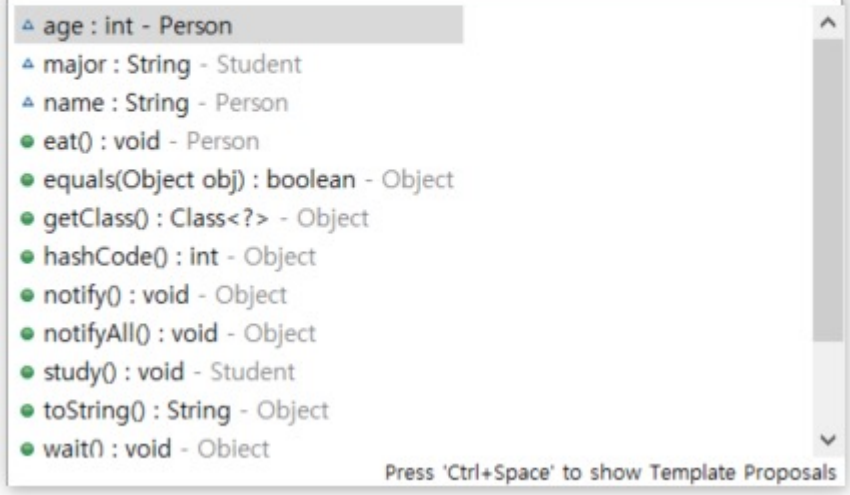
상속(Inheritance)

- 다음 중 다른 부분을 찾으시오.

```
Student st = new Student();
st.
```



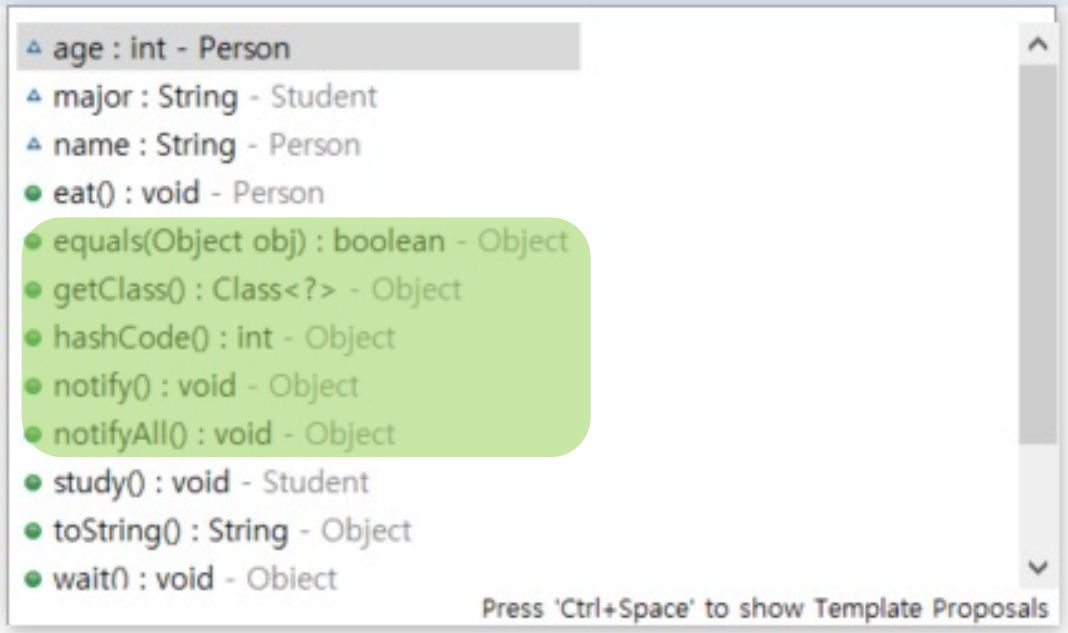

```
Student st = new Student();
st.
```



상속(Inheritance)

- Object??

```
Student st = new Student();

st.
```



상속(Inheritance)

- 1. 확장성, 재 사용성
  - 부모의 생성자와 초기화 블록은 상속 x
- 2. 클래스 선언 시 extends 키워드를 명시
  - 자바는 다중 상속 허용X, 단일 상속 지원
- 3. 관계
  - 부모 (상위, Super) 클래스 : Person
  - 자식 (하위, Sub) 클래스 : Student
- 4. 자식 클래스는 부모 클래스의 멤버변수, 메소드를 자신의 것처럼 사용할 수 있다.

  (단, 접근 제한자에 따라 사용 여부가 달라진다.)
- 5. Object 클래스는 모든 클래스의 조상 클래스
  - 별도의 extends 선언이 없는 클래스는 extends Object가 생략

6. super 키워드

- super를 통해 조상 클래스의 생성자 호출

```java
public class Person {
    String name;
    int age;

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

    public void eat() {
        System.out.println("음식을 먹는다.");
    }
}
```

```java
public class Student extends Person {
    String major;

    public Student(String name, int age, String major) {
        super(name, age);
        this.major = major;
    }

    public void study() {
        System.out.println("공부를 한다.");
    }
}
```

6. super 키워드

- super를 통해 조상 클래스의 메서드 호출

```java
public class Person {
    String name;
    int age;

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

    public void eat() {
        System.out.println("음식을 먹는다.");
    }
}
```

```java
public class Student extends Person {
    String major;

    public Student(String name, int age, String major) {
        super(name, age);
        this.major = major;
    }
    public void study() {
        super.eat();
        System.out.println("공부를 한다.");
    }
}
```

```java
Student st = new Student("김싸피", 28, "컴퓨터공학");

st.study();
```

```
Console
<terminated> Test [Java Application]
음식을 먹는다.
공부를 한다.
```

상속(Inheritance)

7. 오버라이딩 (재정의, overriding)

```java
public class Student extends Person {
    String major;

    public Student(String name, int age, String major) {
        super(name, age);
        this.major = major;
    }

    public void study() {
        super.eat();
        System.out.println("공부를 한다.");
    }

}
```

```java
Student st = new Student("김싸피", 28, "컴퓨터공학");
st.
```

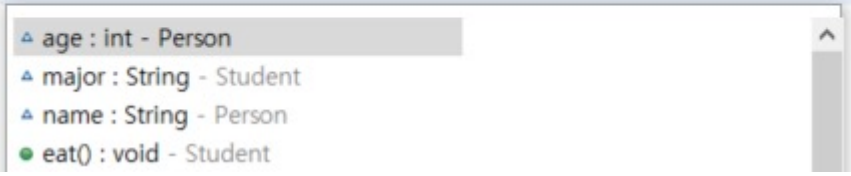

```java
public class Student extends Person {
    String major;

    public Student(String name, int age, String major) {
        super(name, age);
        this.major = major;
    }

    public void study() {
        super.eat();
        System.out.println("공부를 한다.");
    }

    public void eat() {
        System.out.println("지식을 먹는다.");
    }
}
```

```java
Student st = new Student("김싸피", 28, "컴퓨터공학");
st.
```

7. 오버라이딩 (재정의, overriding)

```java
public class Student extends Person {
    String major;

    public Student(String name, int age, String major) {
        super(name, age);
        this.major = major;
    }

    public void study() {
        super.eat();
        System.out.println("공부를 한다.");
    }

    public void eat() {
        System.out.println("지식을 먹는다.");
    }
}
```

```java
Student st = new Student("김싸피", 28, "컴퓨터공학");
st.
```



두 코드의 차이점은??

```java
public class Student extends Person {
    String major;

    public Student(String name, int age, String major) {
        super(name, age);
        this.major = major;
    }

    public void study() {
        super.eat();
        System.out.println("공부를 한다.");
    }
    @Override
    public void eat() {
        System.out.println("지식을 먹는다.");
    }
}
```

```java
Student st = new Student("김싸피", 28, "컴퓨터공학");
st.
```

상속(Inheritance)

7. 오버라이딩 (재정의, overriding)

- 상위 클래스에 선언된 메서드를 자식 클래스에서 재정의 하는 것.
- 메서드의 이름, 반환형, 매개변수 (타입, 개수,순서) 동일 해야 한다.
- 하위 클래스의 접근제어자 범위가 상위 클래스보다 크거나 같아야 한다.
- 조상보다 더 큰 예외를 던질 수 없다.
- 메서드 오버로딩(overloading)과 혼동하지 말 것!!

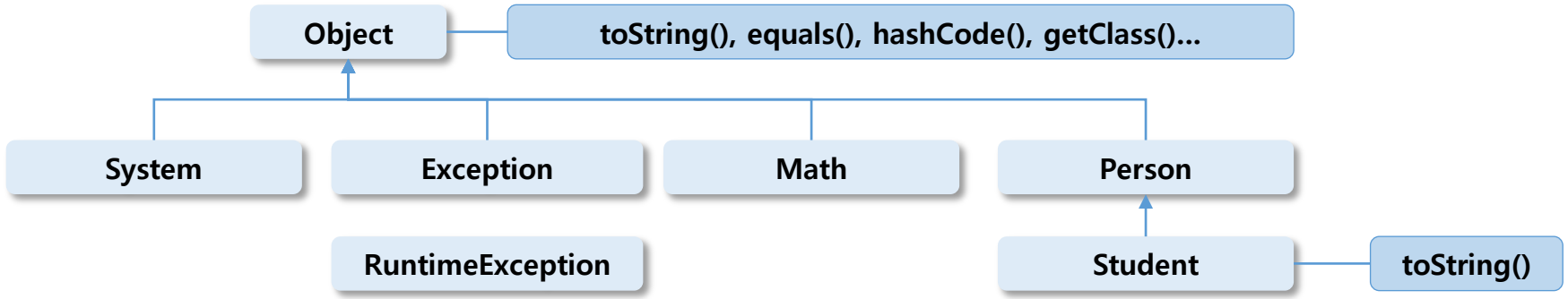
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

Object

Object 클래스

- 가장 최상위 클래스로 모든 클래스의 조상
- Object의 멤버는 모든 클래스의 멤버




```
System.out.println(person.toString());          // Object의 toString 사용

System.out.println(student.toString());         // Student의 toString 사용
```

toString 메서드

- 객체를 문자열로 변경하는 메서드

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

- 정작 궁금한 내용은 주소 값이 아닌 내용이 궁금

```
@Override
public String toString() {
    return “Student [name=" + name + ", age=" + age + ", major=" + major + "]";
}
```

equals 메서드

- 두 객체가 같은지를 비교하는 메서드

```
public boolean equals(Object obj) {
  return (this == obj);
}
```

비교 연산자 == 로 두 객체의 주소 값 비교

- 두 개의 레퍼런스 변수가 같은 객체를 가리키고 있는가?

```
Object obj1 = new Object();
Object obj2 = new Object();
Object obj3 = obj2;
System.out.printf("obj1 == obj2: %b\n", obj1==obj2);
System.out.printf("obj1 equals obj2: %b\n", obj1.equals(obj2));

System.out.printf("obj2 == obj3: %b\n", obj2==obj3);
System.out.printf("obj2 equals obj3: %b\n", obj2.equals(obj3));
```

equals 메서드

- 우리가 비교할 것은 정말 객체의 주소 값인가??
  - 두 객체의 내용을 비교할 수 있도록 equals 메서드 재정의

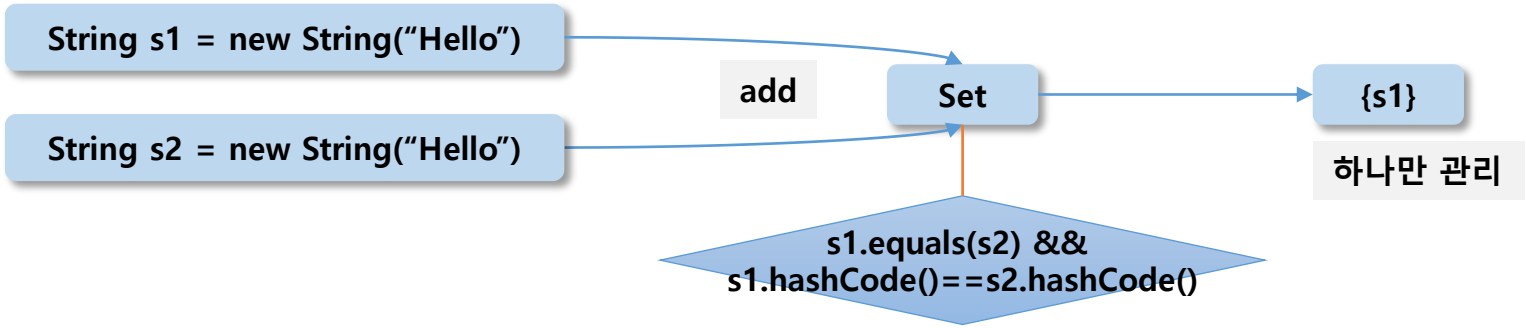
```
private static void testString() {
  String s1 = new String("Hello");
  String s2 = new String("Hello");
  System.out.println((s1 == s2) + " : " + s1.equals(s2));
}

private static void testPhone() {
  Phone p1 = new Phone("0100000000");
  Phone p2 = new Phone("0100000000");
  System.out.println((p1 == p2) + " : " + p1.equals(p2));
}
```

```
class Phone {
  String number = "전화번호";

  public Phone(String number) {
    this.number = number;
  }
/*
  @Override
  public boolean equals(Object obj) {
    if (obj != null && obj instanceof Phone) {
      Phone casted = (Phone) obj;
      return number.equals(casted.number);
    }
    return false;
  }
*/
}
```

  - 객체의 주소 비교 : == 활용
  - 객체의 내용 비교 : equals 재정의

hashCode

- 객체의 해시 코드 : 시스템에서 객체를 구별하기 위해 사용되는 정수값
- HashSet, HashMap 등에서 객체의 동일성을 확인하기 위해 사용



- equals 메서드를 재정의 할 때는 반드시 hashCode도 재정의 할 것
- 미리 작성된 String이나 Number 등에서 재정의 된 hashCode 활용 권장

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

final

final

- 해당 선언이 최종 상태, 결코 수정 될 수 없음.
- final 클래스 : 상속 금지
- final 메소드 : overriding 금지
- final 변수 : 더 이상 값을 바꿀 수 없음 상수화

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미