SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

Java

SAMSUNG SW ACADEMY FOR YOUTH

Programming Language

객체지향 프로그래밍

- 상속
- 다형성

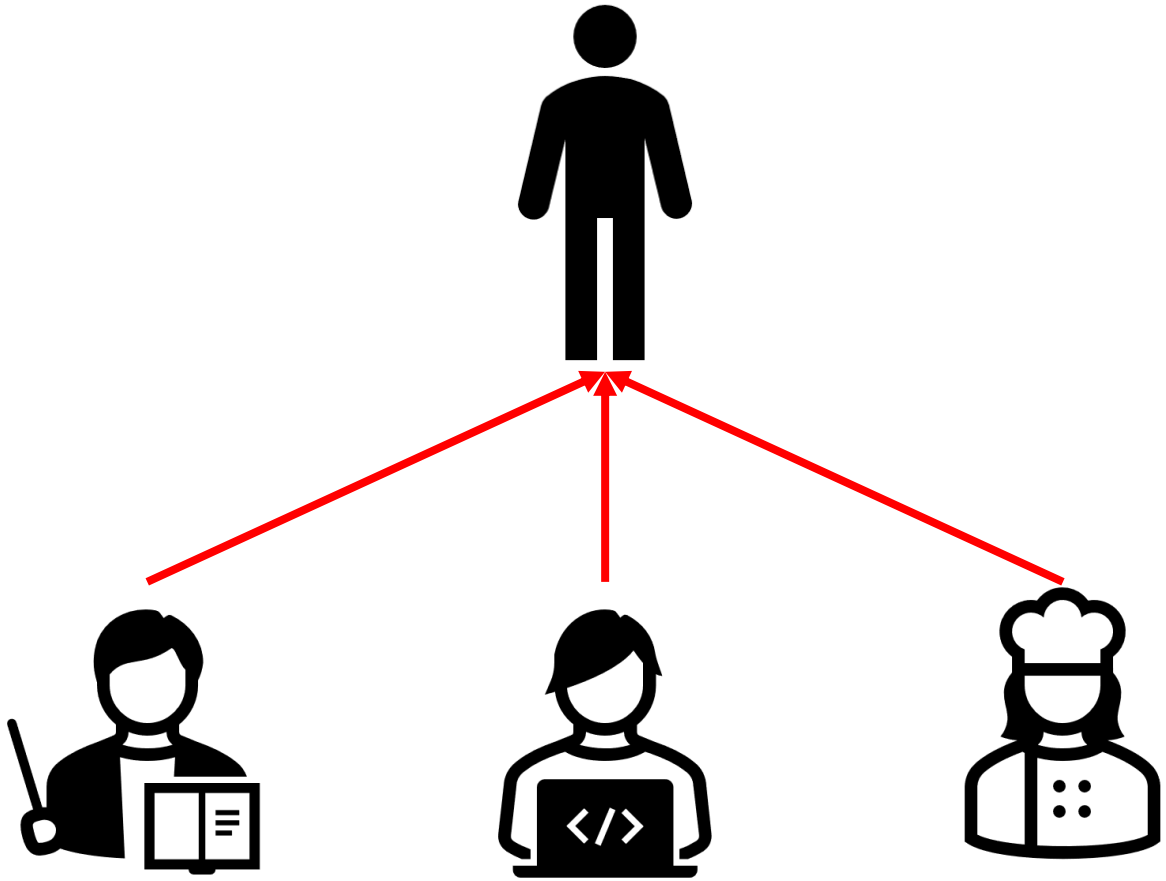
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

상속(Inheritance)

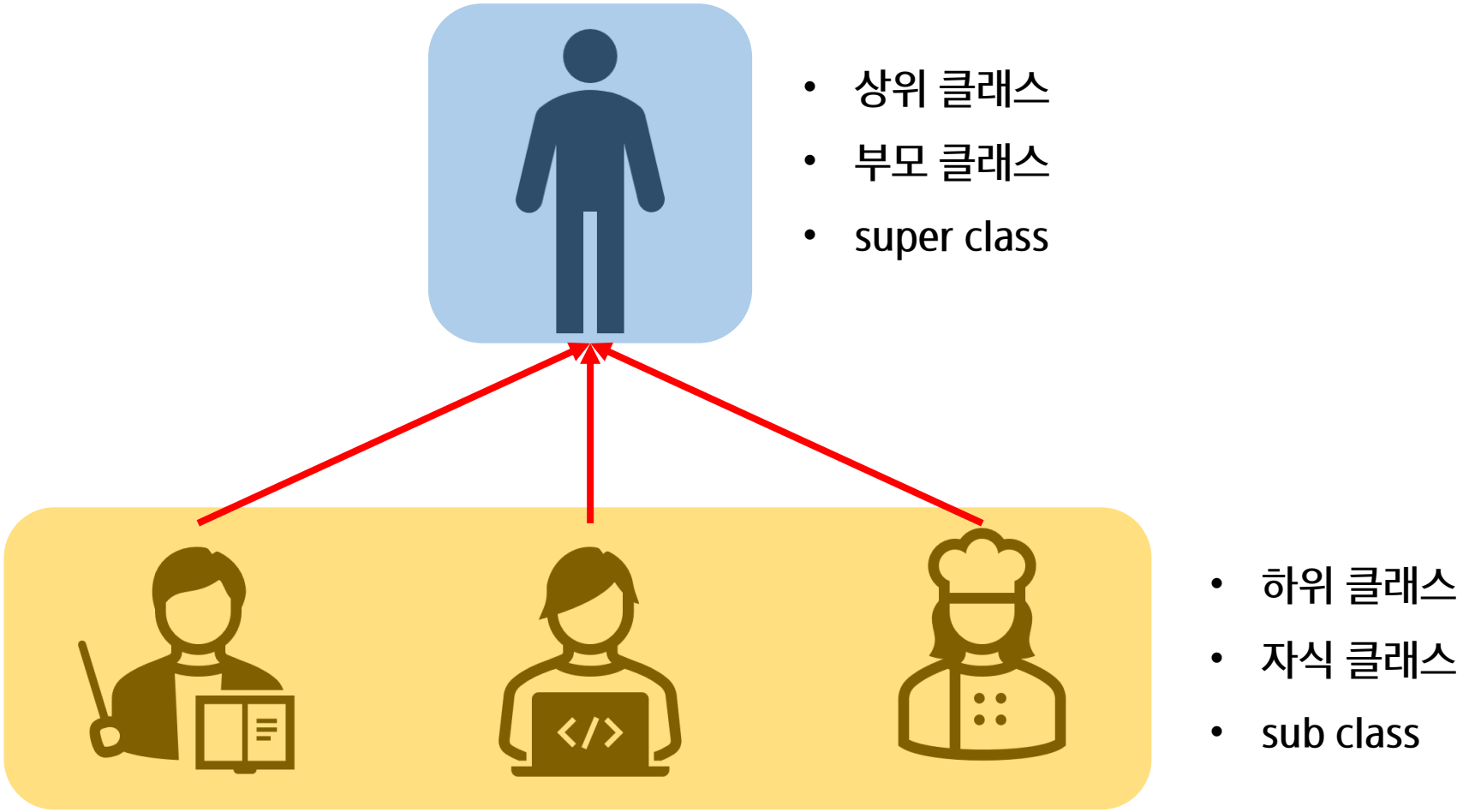
상속(Inheritance)

- 생각 해 봅시다.



상속(Inheritance)

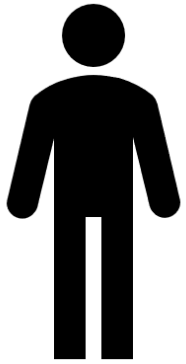
- 어떤 클래스의 특성을 그대로 갖는 새로운 클래스를 정의한 것

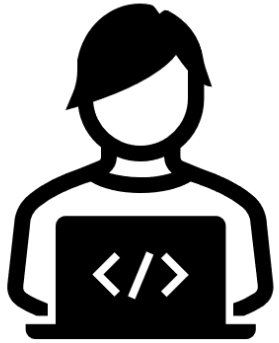


- 상위 클래스
- 부모 클래스
- super class

- 하위 클래스
- 자식 클래스
- sub class

상속(Inheritance)

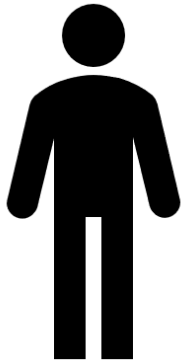


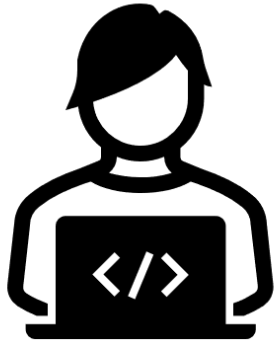


```java
public class Person {
    String name;
    int age;

    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)

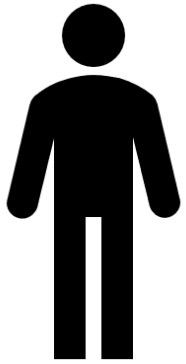


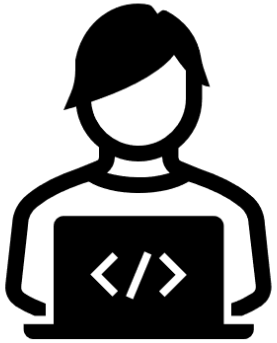


```java
public class Person {
    String name;
    int age;

    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)





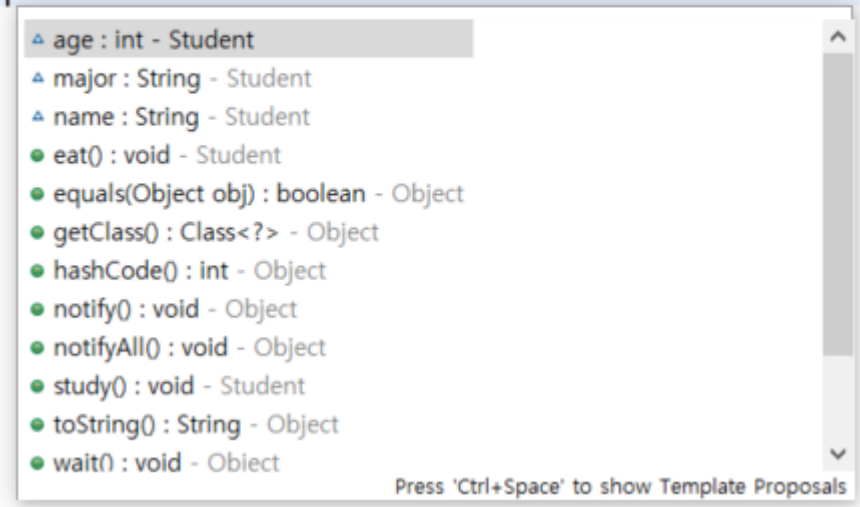
```java
public class Person {
    String name;
    int age;

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

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

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

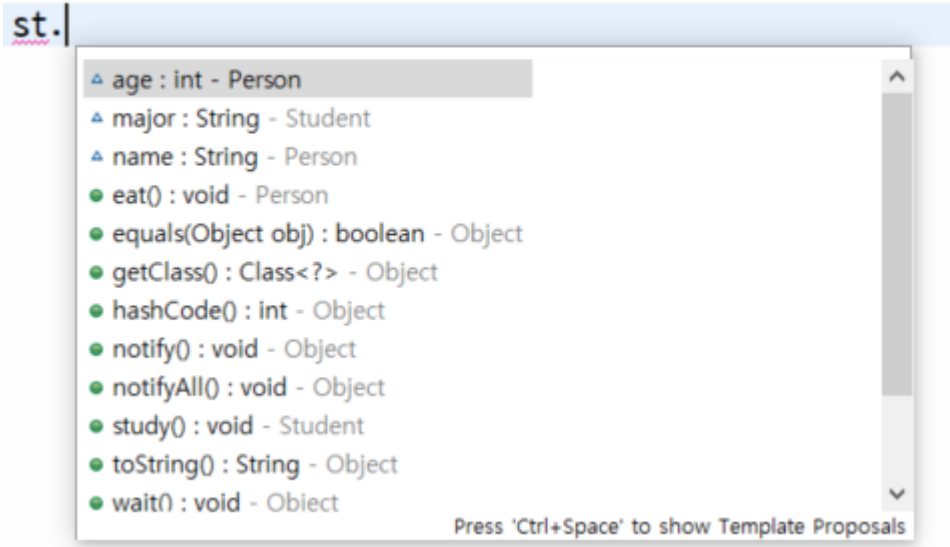
상속(Inheritance)

다음 중 다른 부분을 찾으시오.

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



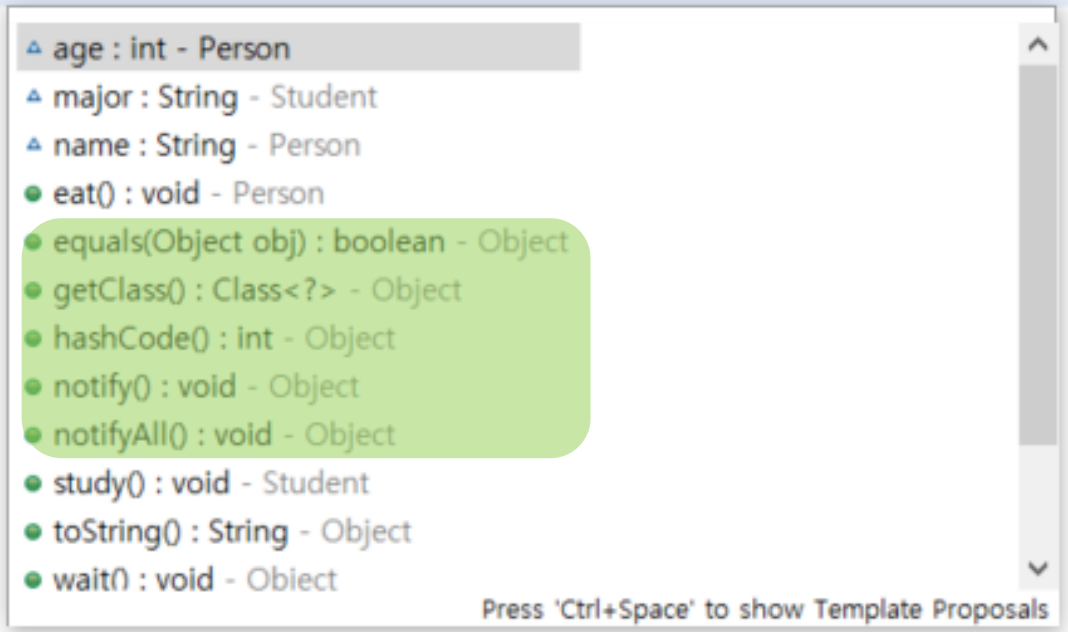

```
Student st = new Student();
st.
```



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.
```

7. 오버라이딩 (재정의, overriding)

```
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.");
    }

}
```

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

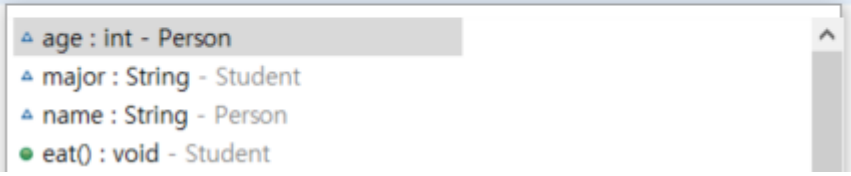

```
public class Student extends Person {
    String major;

    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.");
    }
}
```

```
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.
```

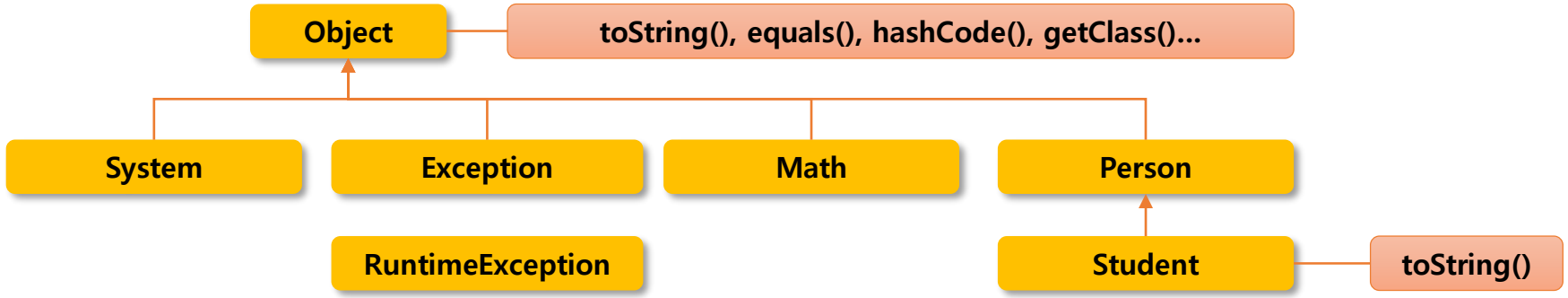
상속(Inheritance)

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!!

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

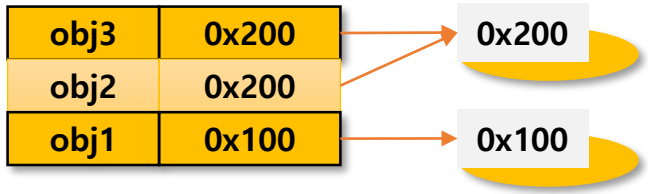
- 두 객체가 같은지를 비교하는 메서드

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

등가비교 연산자 ==로 두 객체의 주소값 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

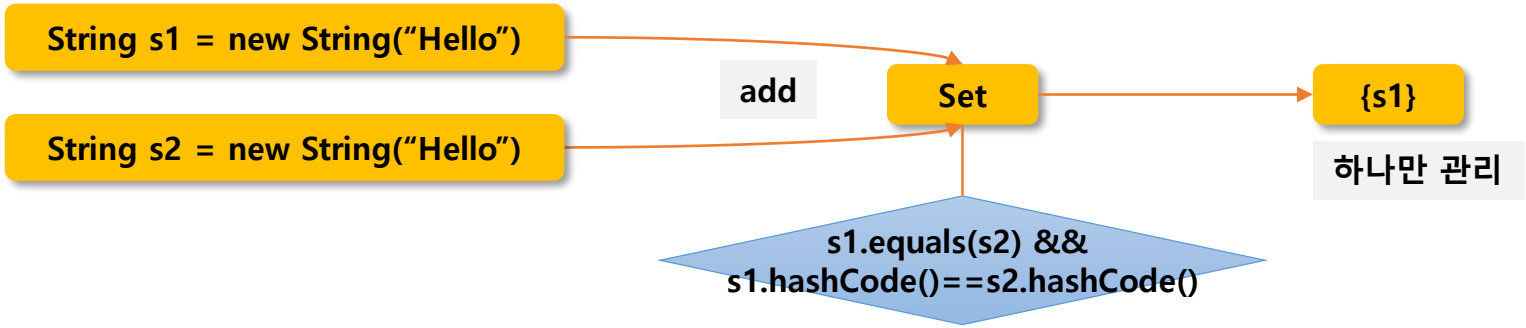
```
private static void testString() {
  String s1 = new String("Hello");
  String s2 = new String("Hello");
  System.out.println((s1 == s2) + " : " + s1.equals(s2));
}

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

- 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

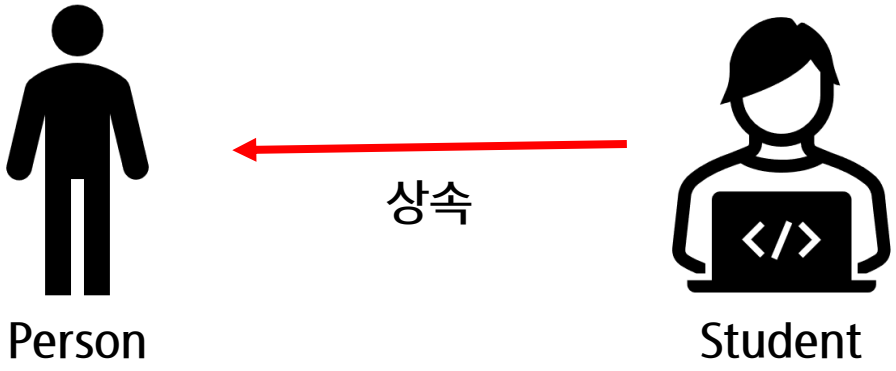
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다형성(Polymorphism)

다형성(Polymorphism)

- 다형성 이란 多(많을 다), 形(형상 형)을 가질 수 있는 성질
- 상속관계에 있을 때 조상 클래스의 타입으로 자식 클래스 객체를 참조할 수 있다.

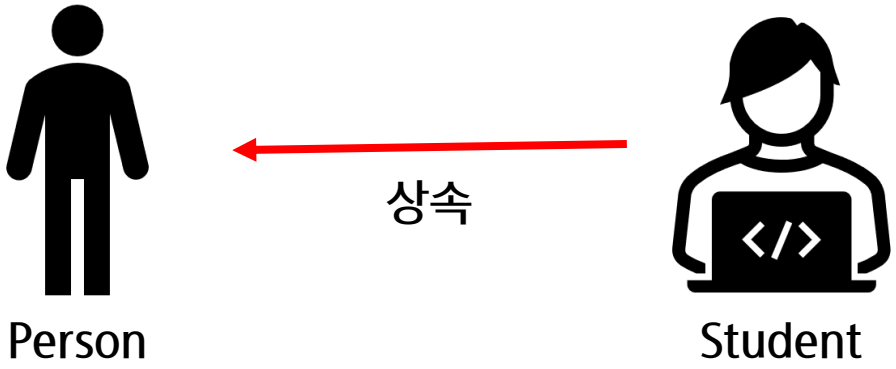


```
Person p = new Student("박싸피", 20, "수학");
```

```
Student st = new Student("이싸피", 25, "경영학");
```

다형성(Polymorphism)

- 다형성 이란 多(많을 다), 形(형상 형)을 가질 수 있는 성질
- 상속관계에 있을 때 조상 클래스의 타입으로 자식 클래스 객체를 참조할 수 있다.

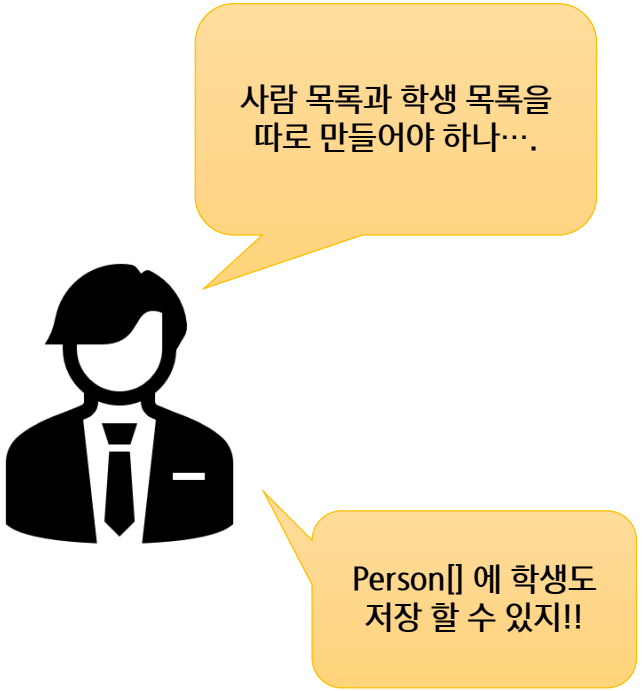


```
Object ob = new Person("최싸피", 33);
```

```
Student st = new Person("정싸피", 27);
```

다형성(Polymorphism)

다형성의 활용 1 - 다른 타입의 객체를 다루는 배열




```
Person[] persons = new Person[3];

persons[0] = new Person();
persons[1] = new Student();
persons[2] = new Student();
```

다형성(Polymorphism)

- 다형성의 활용 2 - 매개변수의 다형성
- 무엇인가를 출력하자.
  - 메서드가 호출되기 위해서는 메서드 이름과 파라미터가 맞아야 한다.
  - public void println(Person p)
  - public void println(Student st)
  - …
- println의 코드

```
public void println(Object x) {
    String s = String.valueOf(x);
    synchronized (this) {
        print(s);
        newLine();
    }
}
```

- 조상을 파라미터로 처리한다면 객체의 타입에 따라 메서드를 만들 필요가 없어진다.

다형성(Polymorphism)

- 다형성의 활용 2 - 매개변수의 다형성
- API 에서 파라미터로 Object를 받는다는 것은 모든 객체를 처리한다는 말

println

```
public void println(Object x)
```

Prints an Object and then terminate the line. This method calls at first String.valueOf(x) to get the printed object's string value, then behaves as though it invokes `print(String)` and then `println()`.

Parameters:

x - The `Object` to be printed.

equals

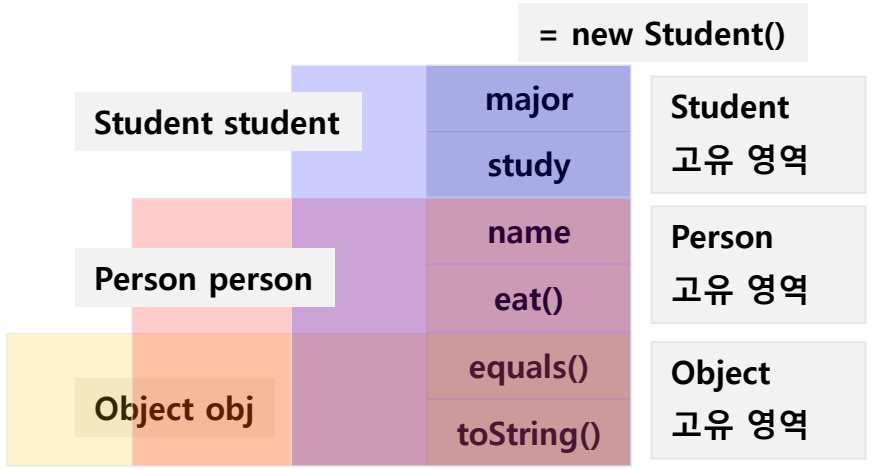
```
public boolean equals(Object obj)
```

Indicates whether some other object is "equal to" this one.

- 필요시 하위 클래스에서 오버라이딩 필요

다형성(Polymorphism)

- 다형성과 참조형 객체의 형 변환
- 메모리에 있는 것과 사용할 수 있는 것의 차이




```
Person p = new Student("박싸피", 20, "수학");
p.
```

```
eat() : void - Person
equals(Object obj) : boolean - Object
getAge() : int - Person
getClass() : Class<?> - Object
getName() : String - Person
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
setAge(int age) : void - Person
setName(String name) : void - Person
toString() : String - Object
wait() : void - Object
Press 'Ctrl+Space' to show Template Proposals
```

- 메모리에 있더라도 참조하는 변수의 타입에 따라 접근할 수 있는 내용이 제한됨.

다형성(Polymorphism)

- 참조형 객체의 형 변환
- 작은 집(child) 에서 큰 집(super) 으로 → 묵시적 캐스팅

```
byte b = 10;
int i = b;
```

```
Person person = new Person();
Object obj = person;
```

  - 자손 타입의 객체를 조상 타입으로 참조 : 형 변환 생략 가능 (조상의 모든 내용은 자식에 포함)

- 큰 집(Super)에서 작은 집(child) 으로 → 명시적 캐스팅

```
int i = 10;
byte b = (byte)i;
```

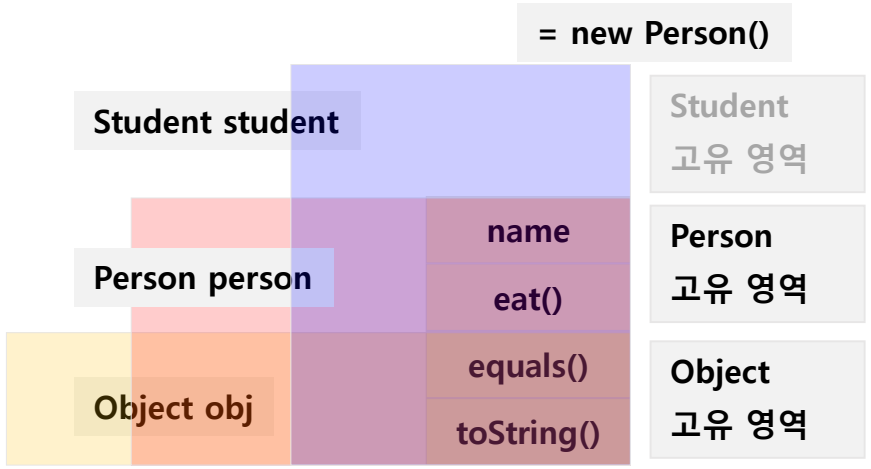
```
Person person = new Student();
Student student = (Student)person;
```

  - 조상 타입을 자손으로 참조 : 형 변환 생략 불가

다형성(Polymorphism)

- 참조형 객체의 형 변환
- 무늬만 Student인 Person

```
Person person = new Person();
Student student = (Student)person;
student.study();
```



- 메모리의 객체는 study() 없음

```
Exception in thread "main" java.lang.ClassCastException:
Person cannot be cast to Student
```

- 조상을 무작정 자손으로 바꿀 수 는 없다.
  - instanceof 연산자 : 실제 메모리에 있는 객체가 특정 클래스의 타입인지 boolean으로 return

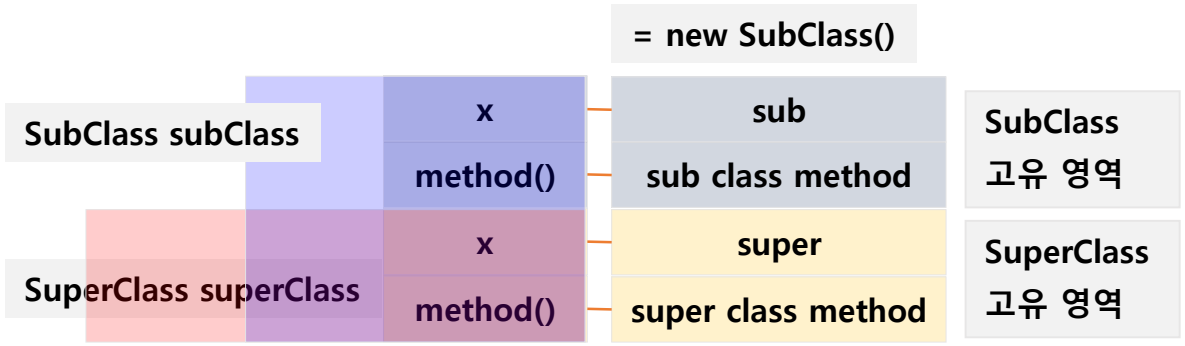
```
Person person = new Person();

if (person instanceof Student) {
      Student student = (Student)person;
      student.study();
}
```

다형성(Polymorphism)

참조 변수의 레벨에 따른 객체의 멤버 연결

```
class SuperClass {
  String x = "super";

  public void method() {
    System.out.println("super class method");
  }
}

class SubClass extends SuperClass {
  String x = "sub";

  @Override
  public void method() {
    System.out.println("sub class method");
  }
}
```

```
public class MemberBindingTest {

  public static void main(String[] args) {
    SubClass subClass = new SubClass();
    System.out.println(subClass.x);
    subClass.method();

    SuperClass superClass = subClass;
    System.out.println(superClass.x);
    superClass.method();
  }
}
```



다형성(Polymorphism)

- 참조 변수의 레벨에 따른 객체의 멤버 연결
- 상속 관계에서 객체의 멤버 변수가 중복 될 때
  - 참조 변수의 타입에 따라 연결이 달라짐

- 상속 관계에서 객체의 메서드가 중복될 때 (메서드가 override 되었을 때)
  - 무조건 자식 클래스의 메서드가 호출됨 → virtual method invocation
  - 최대한 메모리에 생성된 실제 객체에 최적화 된 메서드가 동작한다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미