JAVA Class 상속

프로그램은 사람이 이해하는 코드를 작성.

느려도 꾸준하면 경기에서 이긴다.

작성자 : 홍효상
이메일 : hyomee@naver.com
소스 : https://github.com/hyomee/JAVA_EDU

Content

8. Class 상속

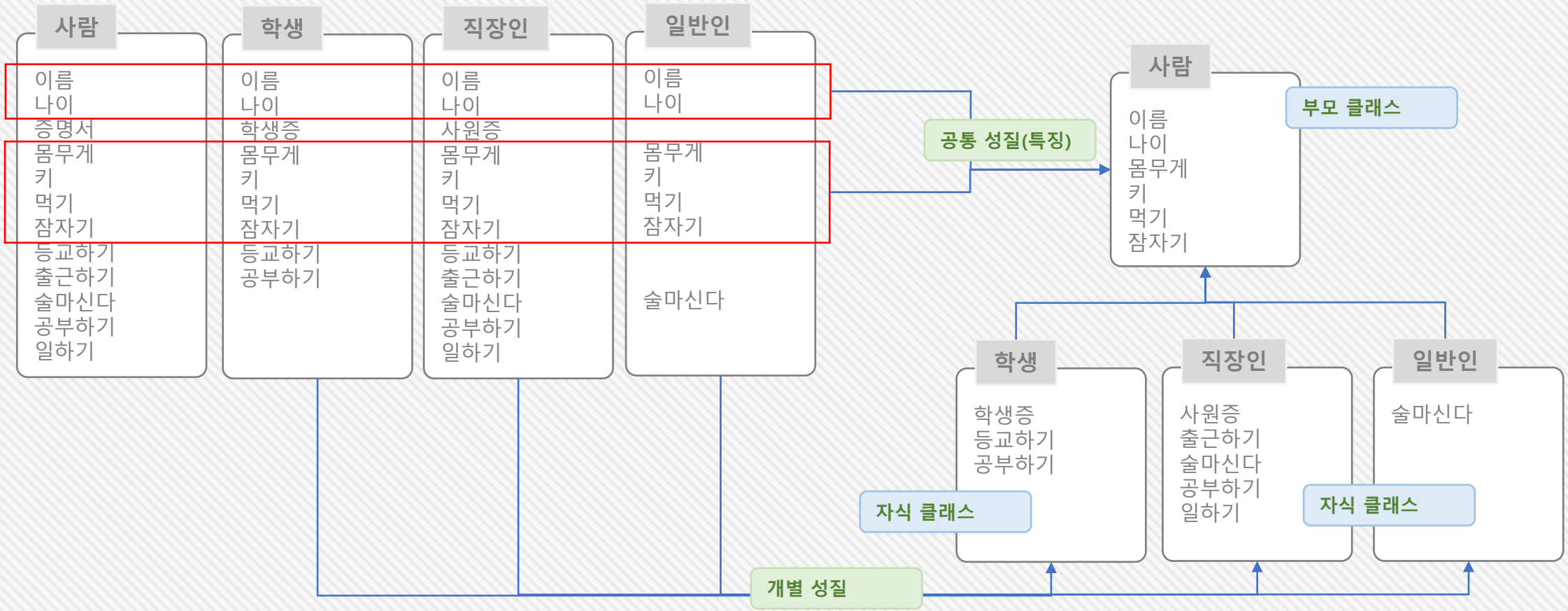
1. 개념
2. Class 상속 (Extends)
3. Abstract Class 상속 (Extends)
4. Interface Class 상속 (Implements)

1. 개념

" 객체 지향 프로그램에서 가장 기본적인 문법 요소 "

클래스 상속

- 부모 클래스를 성질을 자식 클래스가 포함시키는 문법 요소
- 객체(사물)의 공통 적인 성질(특성)을 모아서 클래스를 만들고 독립적인 성질을 각각 클래스로 만들면 이때 공통적인 성질을 부모 클래스라 하고 개별 클래스를 자식 클래스라 한다.

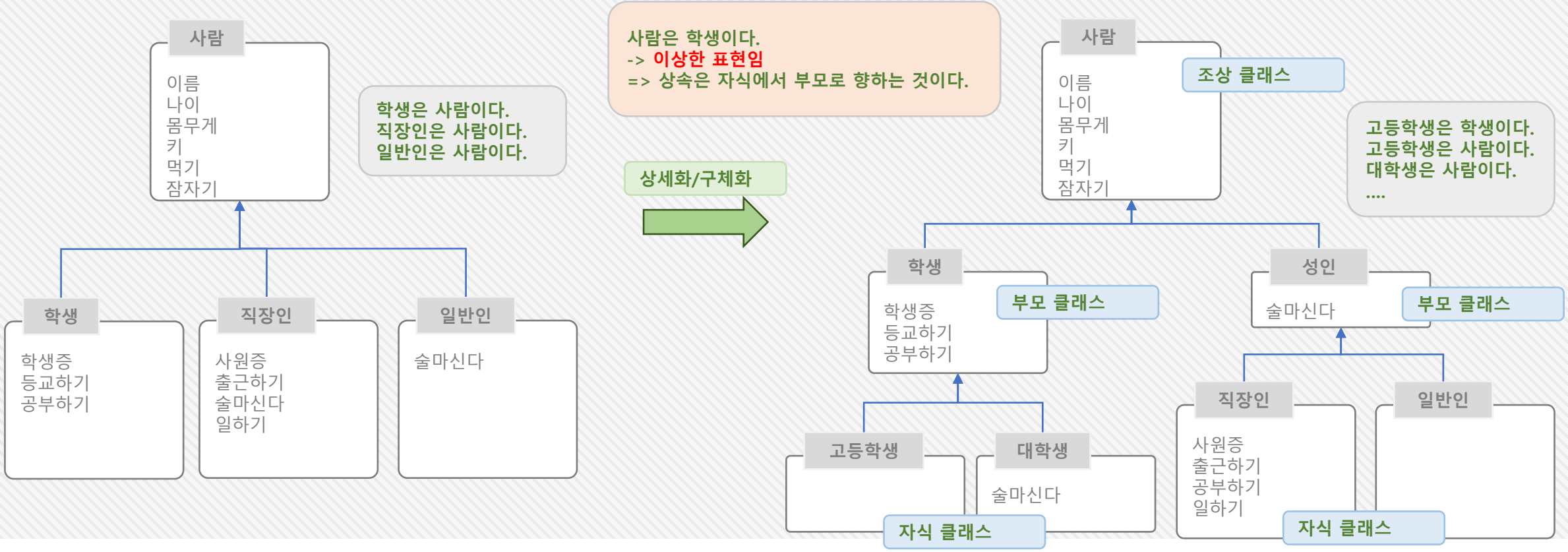


1. 개념

" 상속은 코드 중복 제거와 다형성을 위함이다."

다양성

- 중복 되는 속성(상태)과 행동을 별도로 관리 하여 중복 코드를 제거 한다.
  - 이름, 나이, 몸무게, 키, 먹기, 잠자기
- 학생은 사람이다, 직장인은 사람이다. 직장인은 성인이다. 일반인도 성인이다. 처럼 다양한 표현을 할 수 있는 것을 다양성 이라 한다.

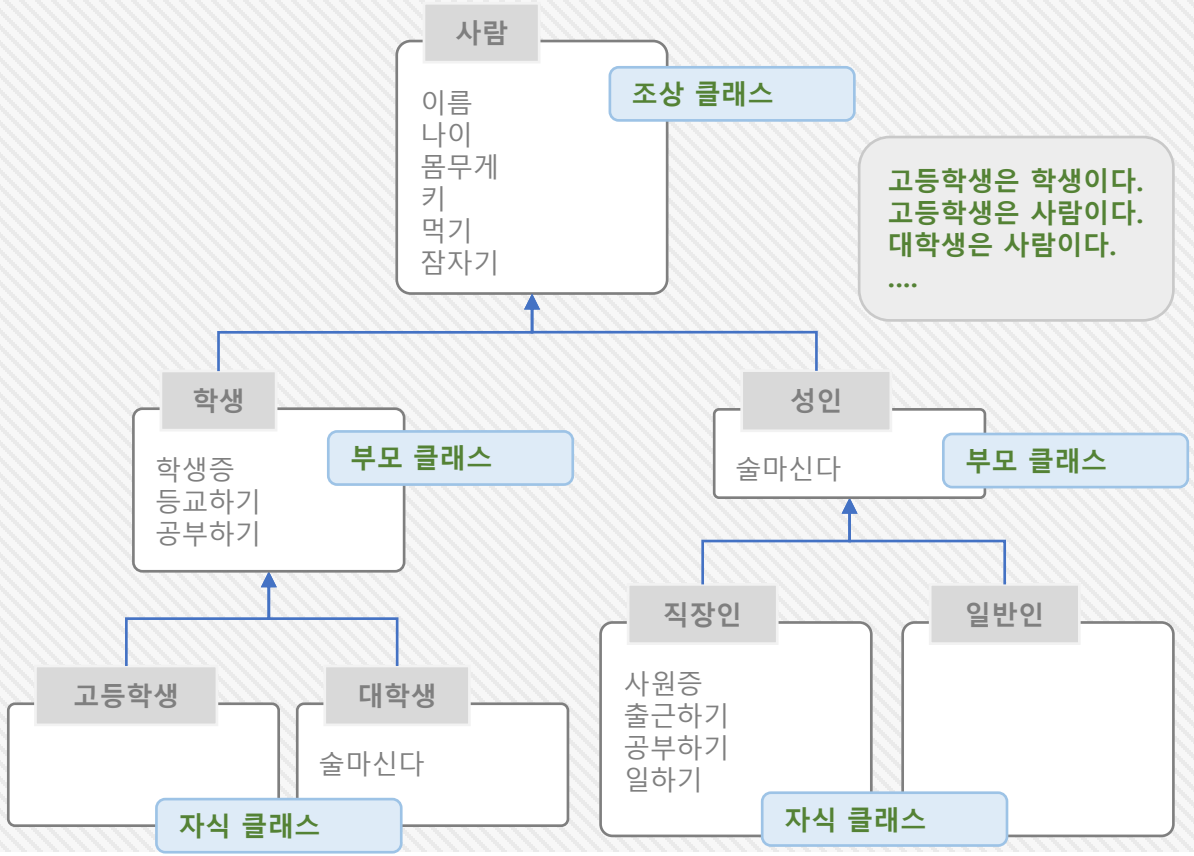


1. 개념

" 상속은 코드 중복 제거와 다형성을 위함이다."

다양성

- 중복 되는 속성(상태)과 행동을 별도로 관리 하여 중복 코드를 제거 한다.
  - 이름, 나이, 몸무게, 키, 먹기, 잠자기
- 학생은 사람이다, 직장인은 사람이다. 직장인은 성인이다. 일반인도 성인이다. 처럼 다양한 표현을 할 수 있는 것을 다양성 이라 한다.



➢ 다양성 표현

- **배열 표현 (상속 되지 않은 상태)**

  - 동일 자료형만 요소로 될 수 있음

```
사람[ ] = { new 사람(), new 사람() };
학생[ ] = { new 학생(), new 학생(), new 학생() };
```

- **배열 표현 (상속 됨)**

  - 부모의 성질(특성)을 모두 받을 수 있음

```
사람[ ] = { new 사람(), new 학생(), new 직장인(), new 일반인() };
```

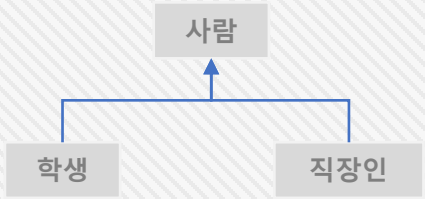
  - 또 다른 표현

```
사람 h01  = new 학생();
사람 h01  = new 직장인();
사람 h01  = new 일반인();
```

1. Class 상속 (Extends)

01. 클래스 생성

```
class 자식클래스 extends 부모클래스 {
  ....
}
```

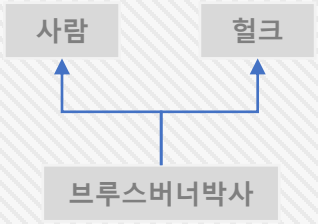



```
class 학생 extends 사람 { ... }
class 직장인 extends 사람  { .... }
```

➢ 다중 상속 -> **자바에서는 허용 하지 않음**

```
class 브루스버너박사 extends 사람, 헐크 { ... }
```

브루스버너박사가 달린다 -> 누가 달리는 것인가 ?
=> 모호성이 발생함




```java
public class Human {
  String name;
  int age;
  float weight;
  float height;

  void eat() {
    System.out.println("먹는다.");
  }
  void sleep() {
    System.out.println("잔다.");
  }
}
```

```java
public class Student extends Human  {
  String studentId;
  void goToSchool() {
    System.out.println("학교에 간다.");
  };
  void study() {
    System.out.println("공부 한다..");
  };
}
```

```java
public class Adult extends Human {
  void drinking() {
    System.out.println("술을 마신다.");
  };
}
```

```java
public class CollegeStudent extends Student{
  void drinking() {
    System.out.println("술을 마신다.");
  };
}
```

```java
public class Worker extends Adult {
  String workerId;
  void goToWork() {
    System.out.println("출근 한다.");
  };
  void study() {
    System.out.println("공부 한다..");
  };
  void doWork() {
    System.out.println("일을 한다.");
  };
}
```

1. Class 상속 (Extends)

02. 클래스 생성 시 메모리 구조

```java
public class Human {
  String name;
  int age;
  float weight;
  float height;

  void eat() {
    System.out.println("먹는다.");
  }
  void sleep() {
    System.out.println("잔다.");
  }
}
```

```java
public class Student extends Human  {
  String studentId;
  void goToSchool() {
    System.out.println("학교에 간다.");
  };
  void study() {
    System.out.println("공부 한다..");
  };
}
```

```java
public class Adult extends Human {
  void drinking() {
    System.out.println("술을 마신다.");
  };
}
```

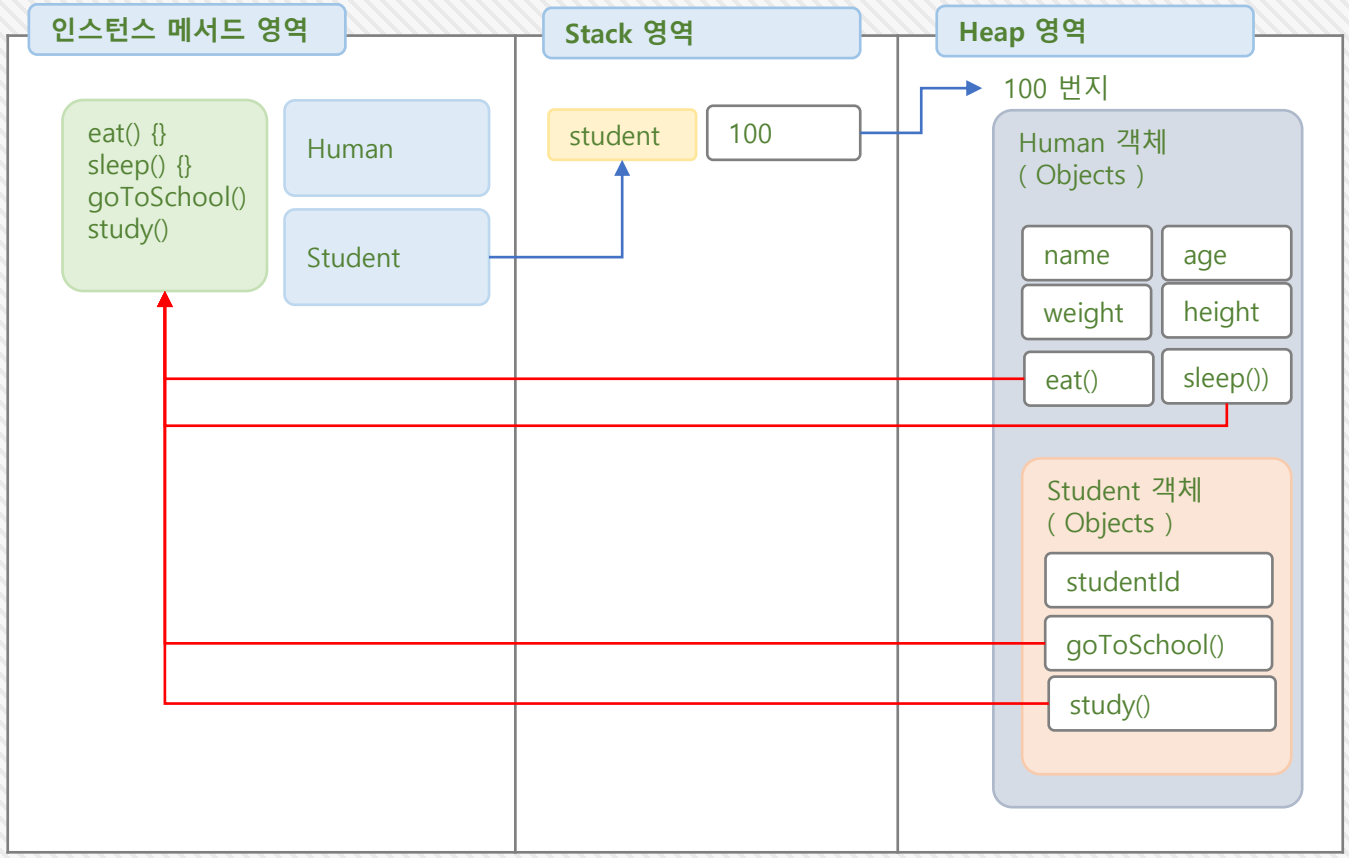
```java
public class CollegeStudent extends Student{
  void drinking() {
    System.out.println("술을 마신다.");
  };
}
```

```java
public class Worker extends Adult {
  String workerId;
  void goToWork() {
    System.out.println("출근 한다.");
  };
  void study() {
    System.out.println("공부 한다..");
  };
  void doWork() {
    System.out.println("일을 한다.");
  };
}
```

➢ 사용

```java
public static void main(String... args) {
  Student student = new Student();

}
```



1. Class 상속 (Extends)

03. 부모 객체에 생성자는 상속 대상이 아니다.

```
public class Student extends Human  {
  String studentId;

  Human() {
  }

  void goToSchool() {
    System.out.println("학교에 간다.");
  };
  void study() {
    System.out.println("공부 한다..");
  };
}
```

Invalid method declaration; return type required
-> 부모 객체의 생성자를 재정의 하려고 해서 나는 오류
-> 자식 객체의 메서드로 사용 하여야 함

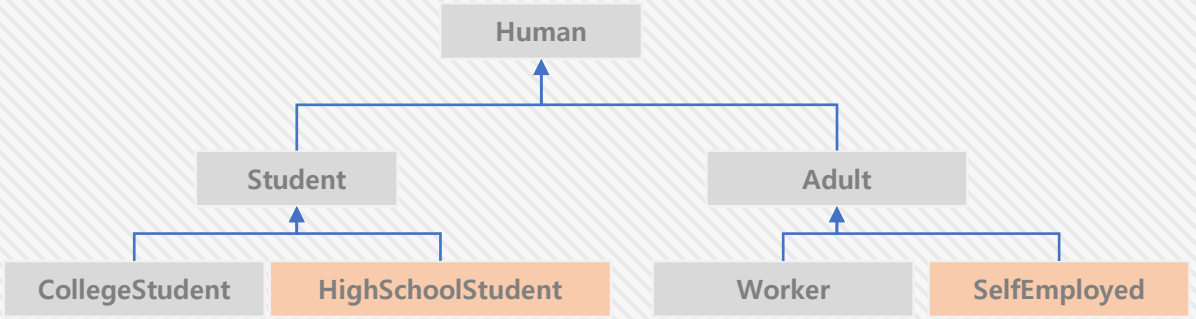
생성자는 리턴 타입이 없음
리턴 타입이 있는 것은 메서드

```
public class Student extends Human  {
  String studentId;

  void Human() {
  }

  void goToSchool() {
    System.out.println("학교에 간다.");
  };
  void study() {
    System.out.println("공부 한다..");
  };
}
```

04. 다양성



public class Human { }

public class Adult extends Human { }
public class Worker extends Adult { }
public class SlefEmpolyed extends Adult { }

public class Student extends Human { }
public class CollegeStudent extends Student { }
public class HighSchoolStudent extends Student { }

- **객체를 다양하게 생성 할 수 있다.**
  -> 중학생, 초등학생 .....

1. Class 상속 (Extends)

05. 객체 생성시 타입

```
public static void main(String... args) {

    Student student = new Student();
    Human human = student;

    Human studentHuman = new Student();
    Student student01 = (Student) studentHuman;

    Student student02 = new Human();

    Student student03 = (Student) new Human();

    Human human01 = new Human();
    Student student04 = (Student) human01;

}
```

Student는 Human를 상속 받아서 Human의 모든 성질(특성)을 사용 할 수 있음
-> **Human, Student 의 모든 속성(멤버), 메서드 사용**

Up Casting : Human human = (Human) student : compiler에 의해서 자동으로 추가
-> Human으로 전화 되었기 때문에 student의 성질(특성)은 사라짐
-> **Human의 멤버, 메서드 만 사용 가능**

Student를 사용 해서 Human으로 생성됨
-> Student의 속성(멤버), 메서드 사용 불가
-> **Human의 멤버, 메서드 만 사용 가능**

Down Casting : 개발자가 직접 해야 함
-> Student 객체를 변환 함
-> **Human, Student 의 모든 속성(멤버), 메서드 사용**

class com.hyomee.extend.Human cannot be cast to class com.hyomee.extend.Student

java.lang.ClassCastException: class com.hyomee.extend.Human cannot be cast to class com.hyomee.extend.Student

Human 객체 생성
-> **Human의 멤버, 메서드 만 사용 가능**

java.lang.ClassCastException: class com.hyomee.extend.Human cannot be cast to class com.hyomee.extend.Student

1. Class 상속 (Extends)

05. 객체 타입 확인 (instanceof)

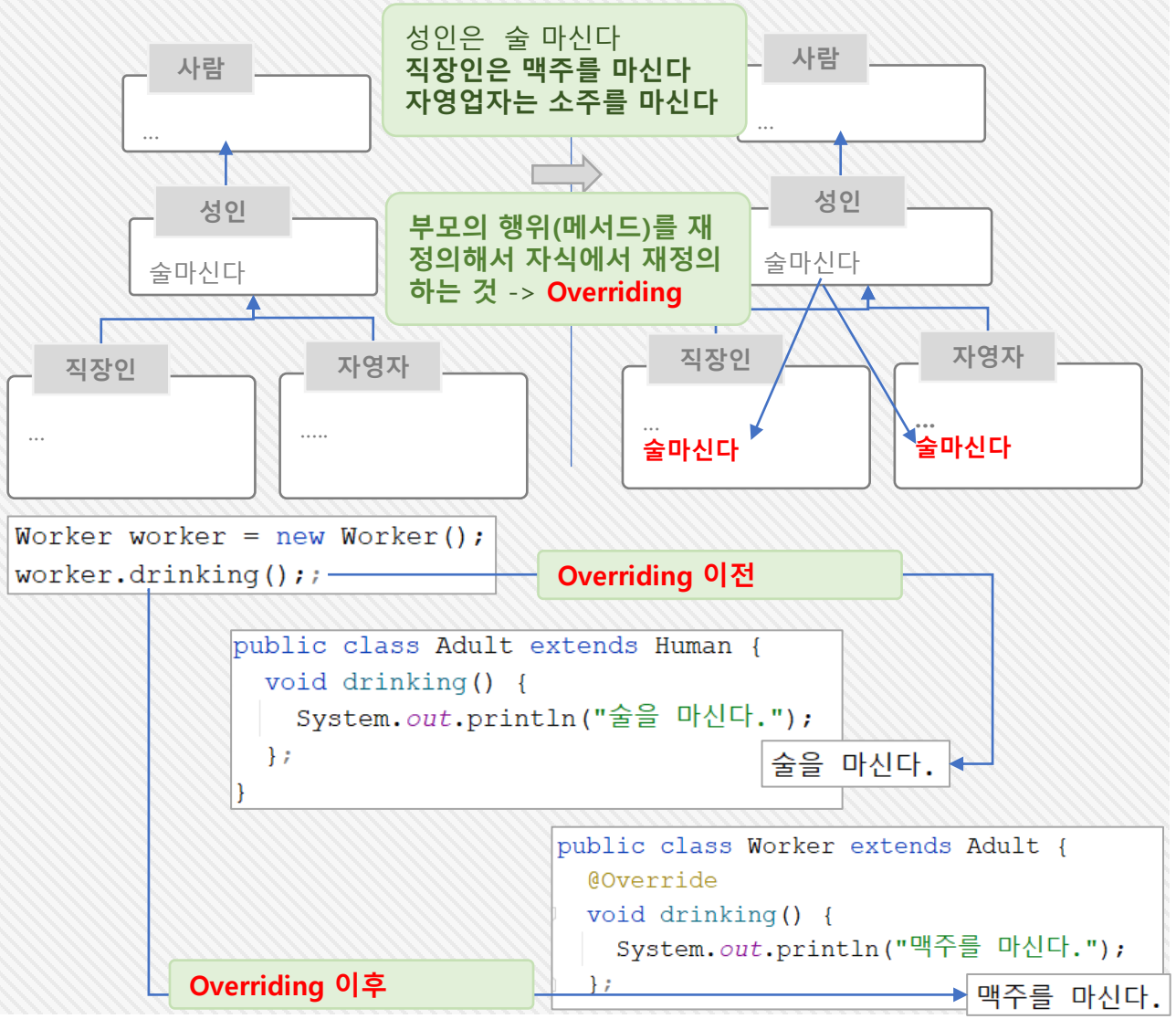
```
Human studentHuman = new Student();
Student student01;
if (studentHuman instanceof Student) {
  System.out.println("human01을 Student로 Castring 가능 합니다.");
  student01 = (Student) studentHuman;
} else {
  System.out.println("human01을 Student로 Castring 불가 합니다.");
}
```

human01을 Student로 Castring 가능 합니다.

```
Human human01 = new Human();
Student student04 ;
if (human01 instanceof Student) {
  System.out.println("human01을 Student로 Castring 가능 합니다.");
  student04 = (Student) human01;
} else {
  System.out.println("human01을 Student로 Castring 불가 합니다.");
}
```

human01을 Student로 Castring 불가 합니다.

06. Method Overriding



1. Class 상속 (Extends)

07. super & this

- super : 부모의 멤버, 메서드를 접근 할 때 사용, private로 선언 것은 접근 할 수 없음
- this : 자기 자신의 멤버, 메서드 접근
- super() : 자식 생성자에서 부모의 생성자에 접근 하기 위해서 사용

```
Worker worker = new Worker();
worker.drinking();;
```

Overriding 이전

```
public class Adult extends Human {
  void drinking() {
    System.out.println("술을 마신다.");
  };
}
```

술을 마신다.

```
public class Worker extends Adult {
  @Override
  void drinking() {
    System.out.println("맥주를 마신다.");
  };
```

맥주를 마신다.

Overriding 이후

super를 이용해서 부모 호출

➢ 술을 마신다를 출력 하고 싶다면

```
public class Worker extends Adult {
  @Override
  void drinking() {
    super.drinking();
    System.out.println("맥주를 마신다.");
  };
```

```
public class Human {
  private String name;
  void setName(String name) {
    this.name = name;
  }
  String getName() {
    return this.name;
  }
```

```
public class Adult extends Human
```

```
public class Worker extends Adult {
  private String workerId;
  void setWorkerId(String workerId) {
    this.workerId = workerId;
  }
  String getWorkerId() {
    return this.workerId;
  }
  void setHumanName(String name) {
    super.setName(name);
  }
```

```
Worker worker = new Worker();
worker.drinking();
worker.setName("홍길동");
System.out.println("Human.name " + worker.getName());
worker.setHumanName("홍당무");
System.out.println("Human.name " + worker.getName());
worker.setWorkerId("ID100001");
System.out.println("Human.name " + worker.getWorkerId());
```

1. Class 상속 (Extends)

08. super()

- super() : 자식 생성자에서 부모의 생성자에 접근 하기 위해서 사용

```java
public class Human {
  private String name;
  Human() { }
  Human(String name) {
    System.out.println("Human Overloading 생성자 생성");
    this.name = name;
  }
```

```java
public class Adult extends Human {
  Adult() {}
  Adult(String name) {
    // Call to 'super()' must be first statement in constructor body
    // System.out.println("Adult OverLoading 생성자 호출");
    super(name);
    System.out.println("Adult OverLoading 생성자 호출");
  }
```

```java
public class Worker extends Adult {
  private String workerId;
  Worker() { }
  Worker(String name) {
    super(name);
    System.out.println("Worker OverLoading 생성자 호출");
  }
```

3

2

1

```java
Worker workerSetName = new Worker("봉선화");
System.out.println("Human.name " + workerSetName.getName());
```

```
Human Overloading 생성자 생성
Adult OverLoading 생성자 호출
Worker OverLoading 생성자 호출
Human.name 봉선화
```

➢ 주의사항

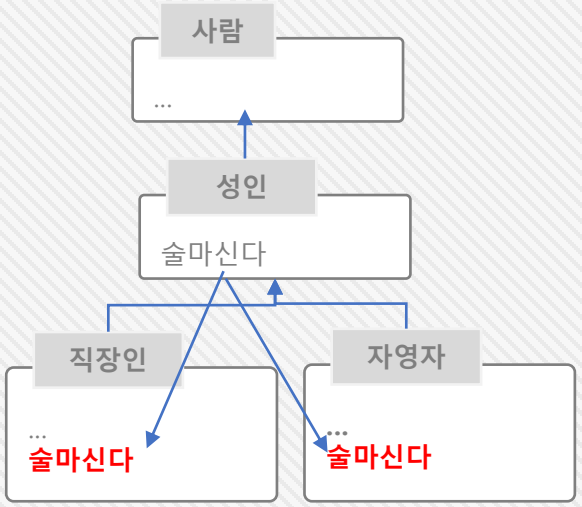
- 생성자를 OverLoading Method를 만들 때는 매개변수 없는 기본 생성자를 작성 해야 함
- super()를 사용 하여 부모 객체를 접근 하는 경우 super()실행 전에 아무 것도 실행 하면 안됨

1. Abatract Class

" 추상 메서드가 1개 이상 포함 하고 있는 클래스 "

Abatract Class

- 추상 메서드가 1개 이상 포함 하고 있는 클래스는 반드시 추상 클래스로 정의 해야 한다.
- 추상 메서드 : **메서드의 본체가 없는 미완성 메서드로** 상속 받은 클래스에서 실체화(구현) 해야 한다.



"성인이 술마신다."를 추상하 하고 **직장인이 술 마시다는 구체화 한다.**

```
public abstract class Adult extends Human {
  Adult() {}

  Adult(String name) {
    super(name);
    System.out.println("Adult OverLoading 생성자 호출");
  }

  abstract void drinking();
}
```

추상 클래스 선언

추상 메서드 있음

추상 메서드 구현

```
public class Worker extends Adult {

  @Override
  void drinking() {
    System.out.println("직장인이 맥주를 마신다.");
  };
```

```
public class AbstractClassMain {
  public static void main(String... args) {
    Worker worker = new Worker();
    worker.drinking();
  }
}
```

직장인이 맥주를 마신다.

1. Interface Class

"시스템 또는 각 객체의 종류에 상관 없이 동일한 필드와 메서드를 제공"

Interface Class

- Interface는 입출력에서 사용 하는 용어로 서로 다른 시스템 간의 연동을 의미 한다.
- Interface는 시스템 또는 각 객체의 종류에 상관 없이 동일한 필드와 메서드를 제공하여 호환성을 제공 하기 위해서 사용 된다.
- 자바 에서 Interface는 객체 관계에서 다른 객체에게 공개할 기능만 추상화 하기 위한 용도로 사용이 된다.
- Class 대신 Interface 키워드를 사용 한다.

01. Interface 생성

```
interface 인터페이스명 {
    public static final 자료형 필드명 = 값;
    public abstract 리턴타입 메서드명();
    default 자료형 메서드(매개변수) { ... )
}
```

생략

```
interface 인터페이스명 {
    자료형 필드명 = 값;
    리턴타입 메서드명();
    default 자료형 메서드(매개변수) { ... )
}
```

```java
public interface Worker{
  // 상속 받는 객체에 자동으로 추가 됨
  String workerId = "F0-00001";
  void goToWorkAndDoWork(String workerId);
  String getWorkerId() ;

  default String defaultWorkerId () {
    return "F0-00000";
  }
}
```

1. Interface Class

02. Interface 상속

- implements를 사용함
- **implements는 다중 상속 가능 -> implements 인터페이스명, 인터페이스명**
- **extends와 같이 사용 하는 경우 extends 클래스명 implements 인터페이스명 순서로 선언 해야 함**

```
class 클래스명 implements 인터페이스명 { .. }
class 클래스명 implements 인터페이스명, 인터페이스명 { .. }
class 클래스명 extends 클래스명 implements 인터페이스명 { .. }
```

상속 받은 객체에 자동
tatic final 로 자동 추가 됨

```java
public interface Worker{
  // 상속 받는 객체에 자동으로 추가 됨
  String workerId = "F0-00001";
  void goToWorkAndDoWork(String workerId);
  String getWorkerId() ;

  default String defaultWorkerId () {
    return "F0-00000";
  }
}
```

```java
public class WorkerImpl implements Worker {
  WorkerVO workerVO = new WorkerVO();

  @Override
  public void goToWorkAndDoWork(String workerId) {
    this.workerVO.setWorkerId(workerId);
    goToWork();
    doWork();
  }

  @Override
  public String getWorkerId() {
    return workerVO.getWorkerId();
  }

  void goToWork() {
     System.out.println(getWorkerId() + " 가 출근 한다.");
  }
```

03. Interface 사용

```
인터페이스명 참조변수 = new 구현클래스();
```

```java
public class InterfaceMain {
  public static void main(String... args) {

    System.out.println(Worker.workerId);

    Worker worker = new WorkerImpl();
    System.out.println(worker.defaultWorkerId());
    worker.goToWorkAndDoWork("FC-00003");
  }
}
```