Chap 07. Class Design – Inheritance

학습목표

1. Inheritance
2. Overriding / Overloading
3. super 키워드

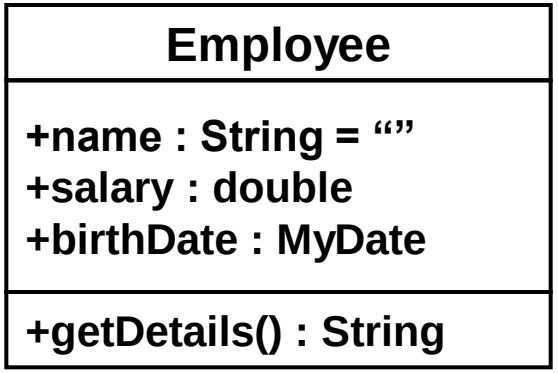
OOP (Object-Oriented Programming) 특징

Encapsulation

Inheritance

Polymorphism

❑ 현재 상황



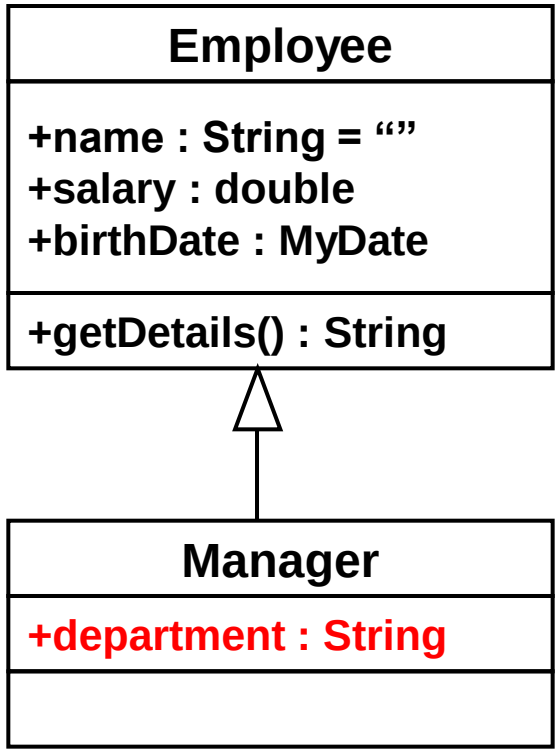

```
public class Employee {

    public String name;

    public double salary;

    public MyDate birthDate;


    public String getDetails() {

        return "Name: " + name + "\n" +

               "Salary: " + salary;

    }

}
```

❑ 새로운 요구

Manager
+name : String = “”   +salary : double   +birthDate : MyDate   +department : String
+getDetails() : String

```
public class Manager {
    public String name;
    public double salary;
    public MyDate birthDate;
    public String department;

    public String getDetails() {
        return "Name: " + name + "\n" +
               "Salary: " + salary + "\n" +
               "Manager of: " + department;
    }
}
```

❑ 해결 : 기존 클래스 이용



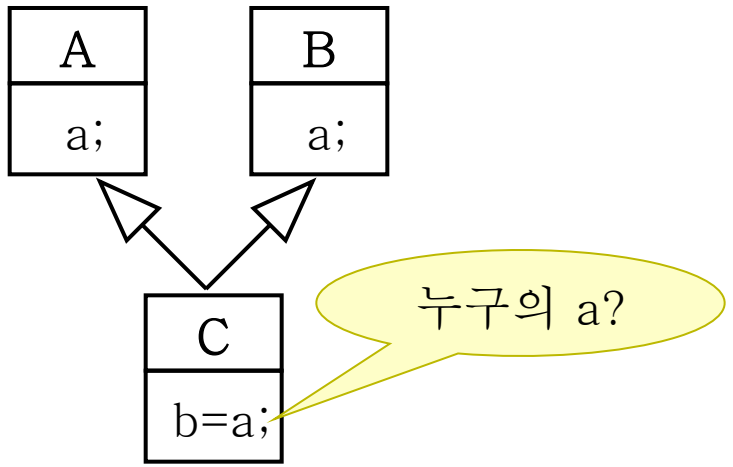

```
public class Employee {
    public String name;
    public double salary;
    public MyDate birthDate;

    public String getDetails() {
        return "Name: " + name + "\n" +
               "Salary: " + salary;
    }
}
```

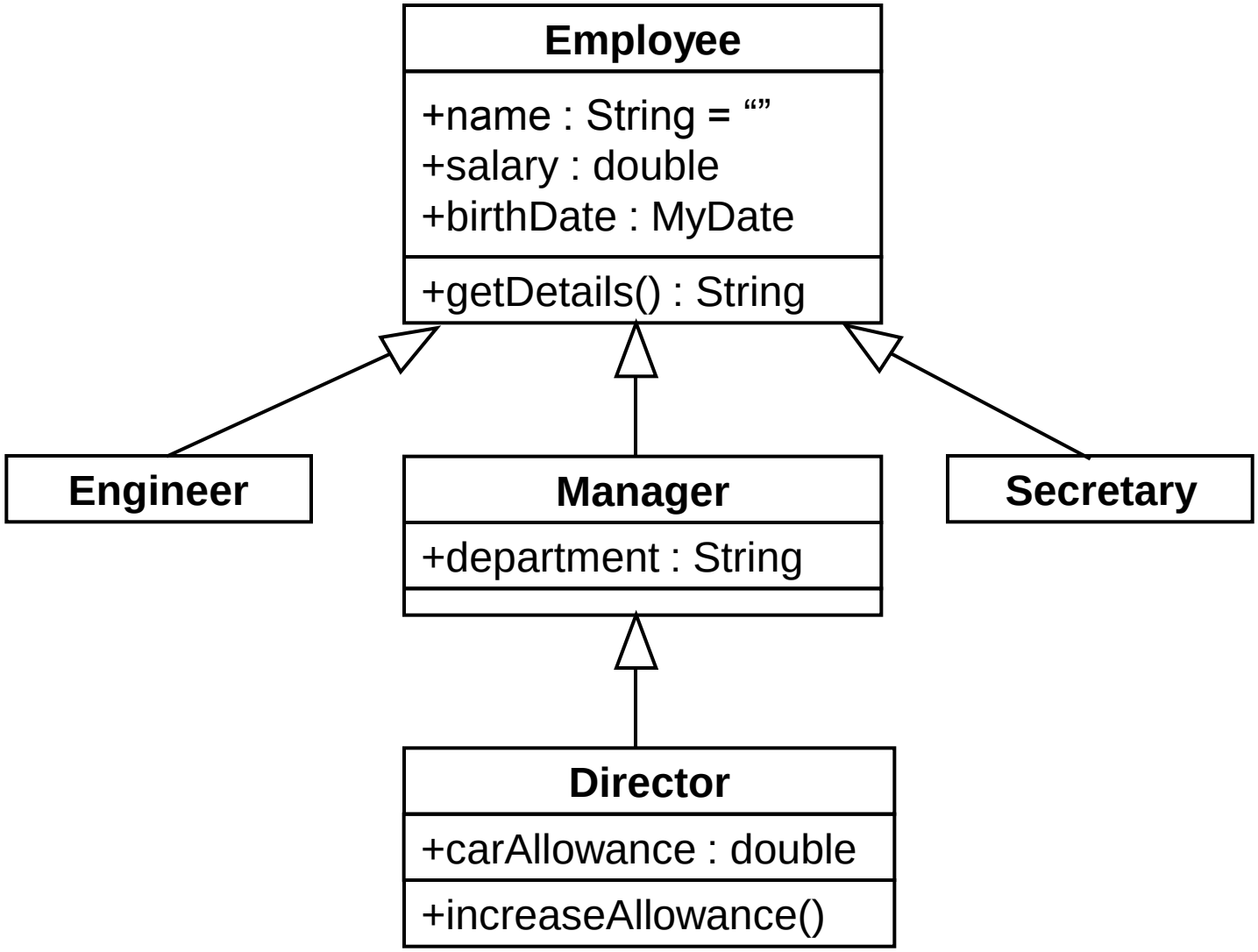
```
public class Manager extends Employee {
    public String department;
}
```

❑ Inheritance (상속)

- 문법 : class *자식클래스* extends *부모클래스*{ … }
- 자바에서의 상속 : 단일 상속만 지원

  즉, 자식은 하나의 부모만 가질 수 있다.
- 단일 상속의 제한점 극복 : interface 이용
- 다중 상속을 허용하지 않는 이유: 코드의 모호성 배제




Employee
+name : String = “”
+salary : double
+birthDate : MyDate
+getDetails() : String
Engineer
Manager
+department : String
Secretary
Director
+carAllowance : double
+increaseAllowance()

접근 제한자

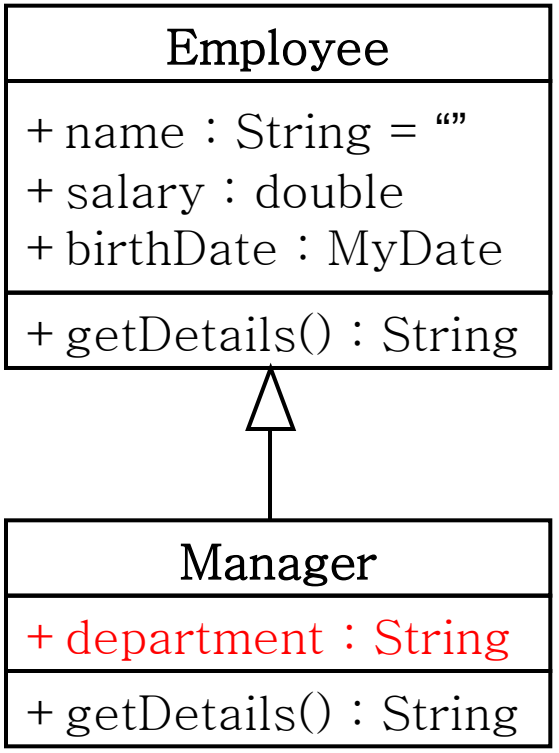
멤버 변수/
멤버 메소드

private	클래스 내부에서만 참조 가능
default	같은 디렉토리(같은 package)내의 다른 클래스에서 참조 가능
protected	default 참조 범위 및 상속 관계에 있을 때 Package관계없이 참조가능
public	어디서나 참조 가능

클래스는 public, *default* 만 가능

Method Overriding (재정의)

❑ 자식 클래스에서 부모 메소드의 기능을 자신의 기능에 맞게 메소드 Body를 새롭게 정의 하는 것.




```
public class Employee {
    public String name;
    public double salary;
    public MyDate birthDate;

    public String getDetails() {
        return "Name: " + name + "\n" +
               "Salary: " + salary;
    }
}
```

```
public class Manager extends Employee {
    public String department;

    public String getDetails(){
        return "Name: " + name + "\n" +
               "Salary: " + salary +
               "\nDepartment: " + department;
    }
}
```

super 키워드

- 부모 객체를 가리킨다.
- 용도 – 부모의 멤버변수 참조 : super.*멤버변수_이름*
  - 부모의 멤버메소드 참조 : super.*멤버메소드_이름()*
  - 부모의 생성자 호출 : super(*파리미터_리스트*)

```
public class Employee {
    private String name;
    private double salary;
    private MyDate birthDate;

    public String getDetails() {
        return "Name: " + name + "\n" +
                "Salary: " + salary;
    }
}
```

```
public class Manager extends Employee {
    private String department;

    public Manager() {
    }

    public String getDetails() {
        return super.getDetails()  +
                "\nDepartment: " + department;
    }
}
```

Inheritance

❑ 상속의 장점

- 비슷한 유형의 코드 재사용
- 검증된 코드를 사용 → 오류의 최소화
- 관련된 여러 클래스들의 공통점 통일화

Method Overloading (다중 정의)

PrintStream Class의 println( ) 메소드

```
public void println( char c ) {  }
public void println( boolean b ) {  }
public void println( int i ) {  }
public void println( long l ) {  }
public void println( double d ) {  }
public void println( float f ) {  }
public void println( String s ) {  }
public void println( Object obj ) {  }
  …
```

```
System.out.println( 10 );
System.out.println( "Java Programming" );
```

- Overloading 의 장점 : 같은 기능에 대해, 같은 메소드 이름을 사용하는 것이 가능하다.

  (일관성 유지)

Constructor Overloading

- 메소드 뿐만 아니라, 생성자도 Overloading 이 가능하다.
- 생성자에서 자신의 다른 생성자 호출 : this() 이용

```
public Employee() {
}
public Employee( String name, double salary ) {
    this( name, salary, null );
}
public Employee( String name, double salary, MyDate birthDate ) {
    this.name = name;
    this.salary = salary;
    this.birthDate = birthDate;
}
```

Constructor – super()

- 생성자는 상속이 안 된다.
- 자식 클래스에서 부모클래스의 멤버변수를 초기화 : super() 이용

```
public Manager() {
}
public Manager( String name, double salary ) {
    this( name, salary, null, null );
}
public Manager( String name, double salary, MyDate birthDate, String department ) {
    super( name, salary, birthDate );
    this.department = department;
}
```

Constructor – 주의 사항

❑ **주의** : this 나 super 키워드가 생성자를 호출하는데 쓰일 경우, 반드시 생성자의 첫 line에 기술되어야 한다. 또한, super(), this() 를 같이 호출하지 못 한다.

```
public Manager() {
}
public Manager( String name, double salary ) {
    super( name, salary ); //error
    this( name, salary, null, null );
}
public Manager( String name, double salary, MyDate birthDate, String department ) {
   this.department = department;   // error !!!
   super( name, salary, birthDate );
 }
```

Constructor – 주의 사항

❑ 주의 : super, this 를 사용하여 생성자를 호출하지 않는 경우, 묵시적으로 super(); 이 호출 된다. 부모 클래스에 default 생성자가 없으면 컴파일 에러

→ 클래스 작성시, 항상 default 생성자를 기술하자!!!

```
public Manager() {
    super();  //묵시적으로 자동 호출
}
public Manager( String name, double salary ) {
    this( name, salary, null, null );
}
public Manager( String name, double salary, MyDate birthDate, String department ) {
    super( name, salary, birthDate );
    this.department = department;
}
```

Overriding vs Overloading

Overriding(재정의)	Overloading(다중정의)
• 메소드를 하위 클래스에서 정의	• 메소드를 같은 클래스에서 정의
• 메소드 이름 동일 • Parameter(개수 및 데이터 타입) 동일 • Return 타입 동일	• 메소드 이름 동일 • Parameter(개수나 데이터 타입) 다름 • Return 타입 다를 수 있음
• 접근 제한자 : 하위 메소드의 접근 범위가 상위 메소드의 접근 범위 보다 넓거나 같아야 한다.	• 접근 제한자 : 관계 없음
• 예외 처리 : 예외 발생시 같은 예외 형식이거나, 더 구체적인 예외 형식이어야 한다.	• 예외 처리 : 관계없음

Method Overriding 규칙 – 접근 제한자

❑ 하위 메소드의 접근 범위가 상위 메소드의 접근 범위 보다 넓거나 같아야 한다.

```
public class Example {
    public static void main(String[] args) {
        SuperClass superClass = new SubClass();

        //Runtime시 타입이 private임. 호출 되어질 수 없음.
        superClass.method();
    }
}
class SuperClass {
    public void method() {
    }
}
class SubClass extends SuperClass {
    private void method() { //컴파일 오류가 안난다고 가정
    }
}
```