# Lecture 11

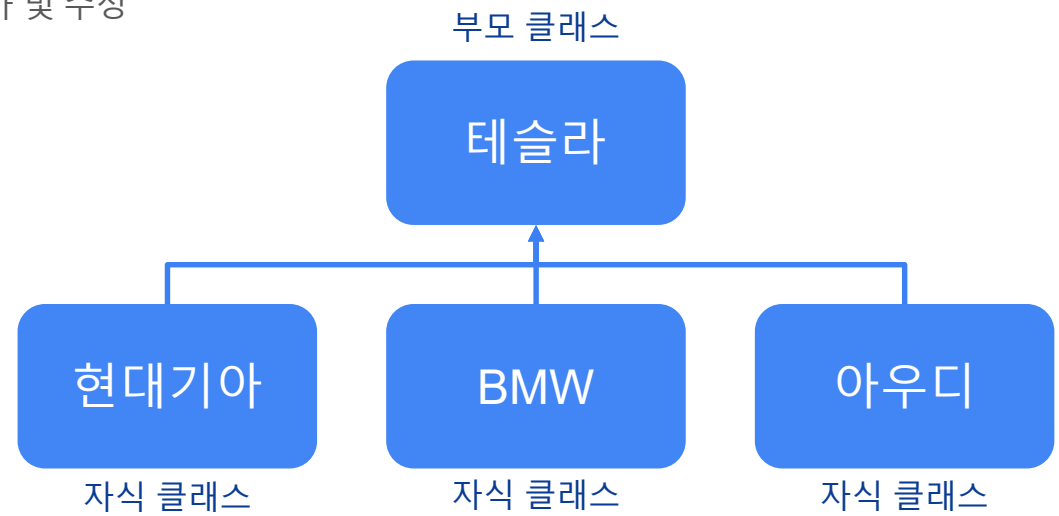
## 상속과 오버라이딩

# 목차

# 상속의 정의와 필요성

만약, 기존의 클래스에서 추가적으로 확장을 하고 싶다면?

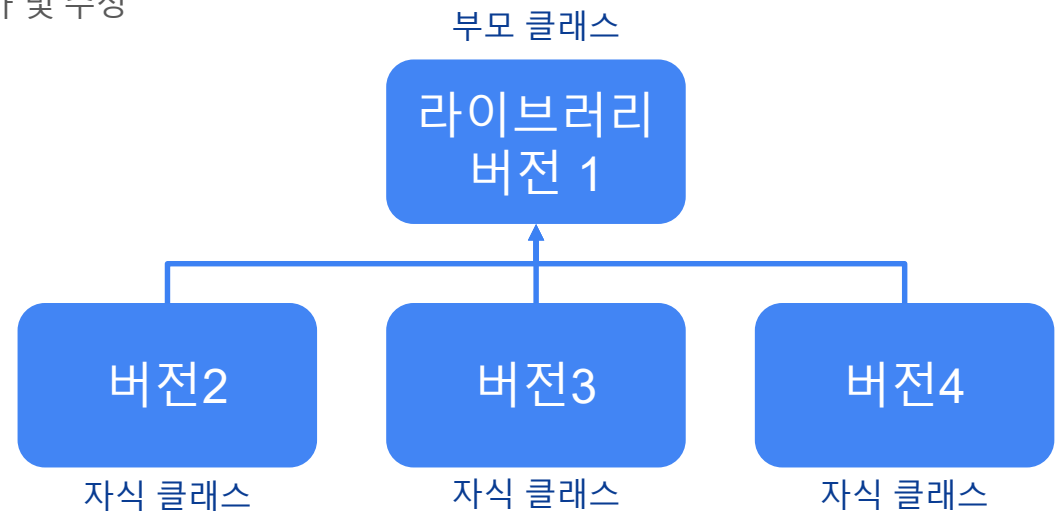
- 멤버변수 추가 및 수정
- 멤버함수 추가 및 수정



# 상속의 정의와 필요성

만약, 기존의 클래스에서 추가적으로 확장을 하고 싶다면?

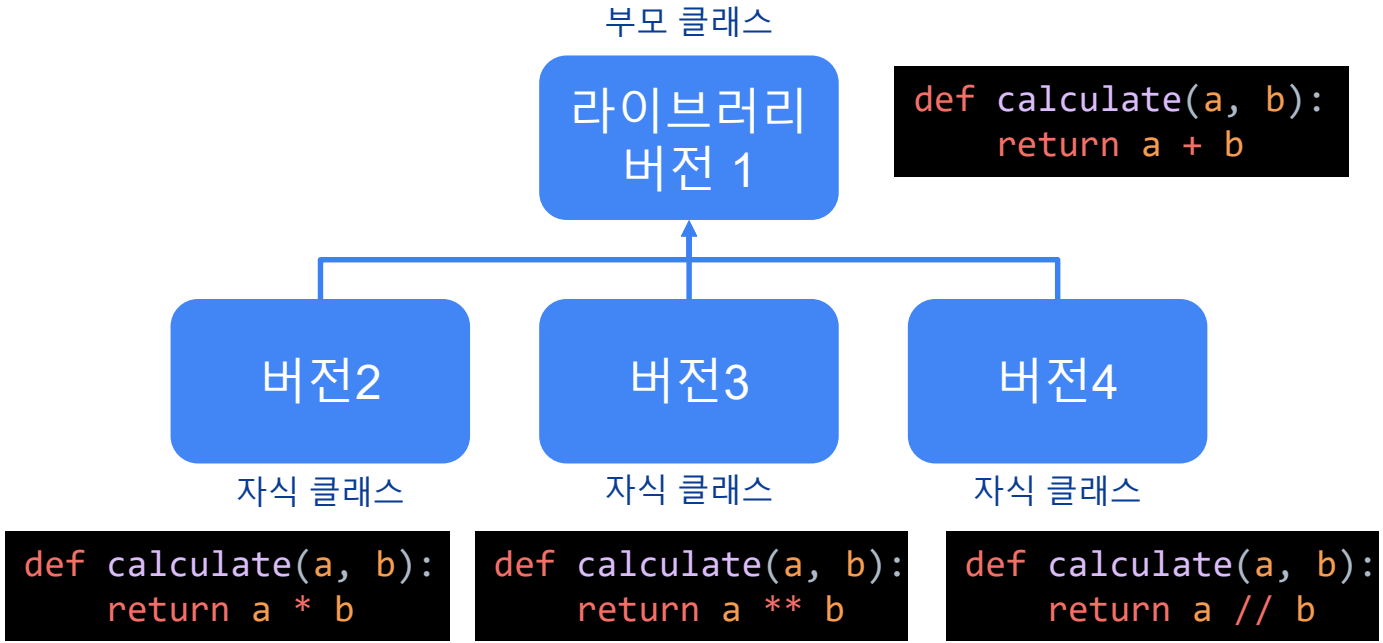
- 멤버변수 추가 및 수정
- 멤버함수 추가 및 수정



# 오버라이딩에 대한 이해

오버라이딩은 무엇이고 왜 필요할까?

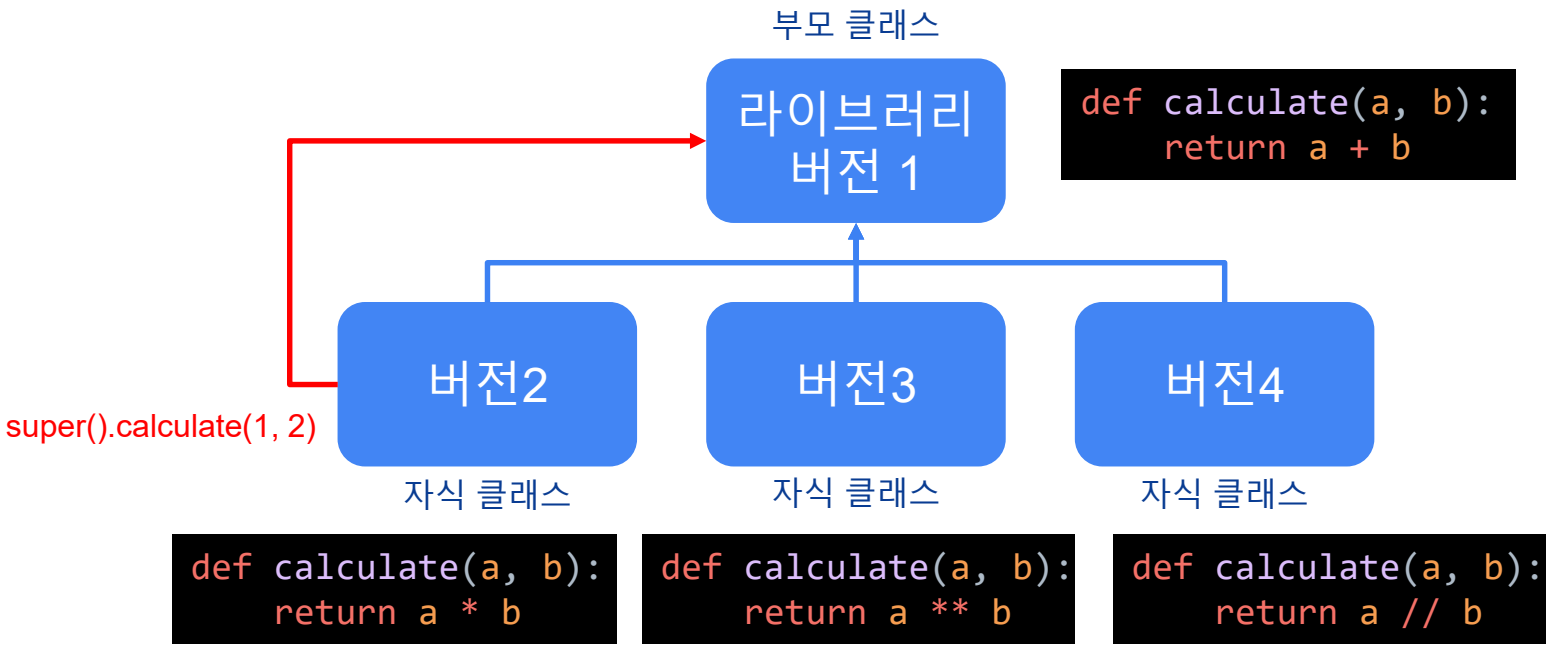
- 부모 클래스에 기존에 존재하는 함수를 변경하고 싶을 때!
- 부모 클래스를 직접 수정하지 않고 변경하고 싶을 때!



# super()에 대한 이해

super()은 무엇이고 왜 필요할까?

❖ 자식클래스에서 부모클래스의 멤버함수를 사용하고 싶을 때! (예) super().멤버함수이름(입력변수)



# 상속 구현 예제

라이브러리 버전1을 상속하는 라이브러리 버전2를 만들어보자!

예제1

```
# 라이브러리 버전1 클래스
class Library_v1:
    pass

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    pass

# 라이브러리 버전2의 인스턴스 생성
instance_v2 = Library_v2()
```

# 상속 구현 예제

라이브러리 버전1을 상속하는 라이브러리 버전2를 만들어보자!

예제1

```
# 라이브러리 버전1 클래스
class Library_v1:
    pass

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    pass

# 라이브러리 버전2의 인스턴스 생성
instance_v2 = Library_v2()
```

괄호안에 상속할 부모클래스를 작성

# 상속 구현 예제

라이브러리 버전1을 상속하는 라이브러리 버전2를 만들어보자!

예제1

```
# 라이브러리 버전1 클래스
class Library_v1:
    pass

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    pass

# 라이브러리 버전2의 인스턴스 생성
instance_v2 = Library_v2()
```

클래스 인스턴스 생성과 동일

# 멤버함수 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제2

```
# 라이브러리 버전1 클래스
class Library_v1:
    def calculate(self, a, b):
        return a + b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def calculate(self, a, b):
        return a * b

# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1()
instance_v2 = Library_v2()

print(instance_v1.calculate(3, 2)) # 출력: 5
print(instance_v2.calculate(3, 2)) # 출력: 6
```

# 멤버함수 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제2

```
# 라이브러리 버전1 클래스
class Library_v1:
    def calculate(self, a, b):
        return a + b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def calculate(self, a, b):
        return a * b

# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1()
instance_v2 = Library_v2()

print(instance_v1.calculate(3, 2)) # 출력: 5
print(instance_v2.calculate(3, 2)) # 출력: 6
```

오버라이딩 (함수 변경)

# 멤버함수 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제3

```
# 라이브러리 버전1 클래스
class Library_v1:
    def calculate(self, a, b):
        return a + b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def calculate(self, x, y, z):
        return x * y * z

# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1()
instance_v2 = Library_v2()

print(instance_v1.calculate(3, 2)) # 출력: 5
print(instance_v2.calculate(3, 4, 5)) # 출력: 60
```

오버라이딩 시에 변수명과 변수의 개수 변경 가능

# 멤버함수 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제4

```
# 라이브러리 버전1 클래스
class Library_v1:
    def calculate(self, a, b):
        return a + b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def calculate(self, x, y, z):
        w = x + y   부모클래스의 멤버함수 이용
        return super().calculate(w, z)

# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1()
instance_v2 = Library_v2()

print(instance_v1.calculate(3, 2)) # 출력: 5
print(instance_v2.calculate(3, 4, 5)) # 출력: 12
```

# __init__ 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제5

```
# 라이브러리 버전1 클래스
class Library_v1:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def calculate(self):
        return self.a + self.b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def __init__(self, x, y):
        super().__init__(x, y)

    def calculate(self):
        return self.a * self.b
```

```
# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1(3, 2)
instance_v2 = Library_v2(3, 4)

print(instance_v1.calculate()) # 출력: 5
print(instance_v2.calculate()) # 출력: 12
```

# __init__ 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제5

```
# 라이브러리 버전1 클래스
class Library_v1:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def calculate(self):
        return self.a + self.b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def __init__(self, x, y):
        super().__init__(x, y)

    def calculate(self):
        return self.a * self.b
```

```
# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1(3, 2)
instance_v2 = Library_v2(3, 4)

print(instance_v1.calculate()) # 출력: 5
print(instance_v2.calculate()) # 출력: 12
```

초기화 함수도 오버라이딩 가능

# __init__ 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제5

```
# 라이브러리 버전1 클래스
class Library_v1:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def calculate(self):
        return self.a + self.b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def __init__(self, x, y):
        super().__init__(x, y)

    def calculate(self):
        return self.a * self.b
```

```
# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1(3, 2)
instance_v2 = Library_v2(3, 4)

print(instance_v1.calculate()) # 출력: 5
print(instance_v2.calculate()) # 출력: 12
```

부모 클래스의 초기화 함수 사용

# __init__ 오버라이딩과 super() 구현예제

라이브러리 버전1의 멤버함수를 직접 수정하지 않고 상속으로 버전 업데이트를 해보자

예제5

```
# 라이브러리 버전1 클래스
class Library_v1:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def calculate(self):
        return self.a + self.b

# 라이브러리 버전2 클래스
class Library_v2(Library_v1):
    def __init__(self, x, y):
        super().__init__(x, y)

    def calculate(self):
        return self.a * self.b
```

```
# 라이브러리 버전1과 버전1을 상속하는 버전2
instance_v1 = Library_v1(3, 2)
instance_v2 = Library_v2(3, 4)
                                    __init__ 의 입력값
print(instance_v1.calculate()) # 출력: 5
print(instance_v2.calculate()) # 출력: 12
```

# 빌트인 라이브러리 함수 기능 추가하기 (+Magic method)

빌트인 라이브러리 함수에 기능을 추가해보자!

```
a = int(3.5)
print(a) # 출력: 3
```

사실, int 클래스의 인스턴스가 a

예제6

```
# int 클래스 버전 업데이트
class int_v2(int):
    def square(self):
        return self ** 2

# 인스턴스 생성
a = int_v2(3.5)

print(a) # 출력: 3
print(a.square()) # 출력: 9
```

square 함수 추가

# 빌트인 라이브러리 함수 기능 추가하기 (+Magic method)

빌트인 라이브러리 함수에 기능을 추가해보자!

예제7

```
# int 클래스 버전 업데이트 하기 위해 생성한 클래스
class int_v3(int):
    # print 할 때 출력되는 magic method (오버라이딩)
    def __repr__(self):
        return str(self * 100)

    # 클래스를 괄호로 호출할 때 나오는 magic method
    def __call__(self):
        # 부모 클래스의 원래 __repr__ 함수 호출
        return super().__repr__()

# int 클래스 버전3의 인스턴스 생성
a = int_v3(3.5)

print(a) # 출력: 300
print(a()) # 출력: 3
```

**Magic Method**

**__init__ : 클래스 초기화 함수**

**__repr__ : print 할 때 출력되는 함수**

**__call__ : 인스턴스를 괄호를 이용해 호출**

# 빌트인 라이브러리 함수 기능 추가하기 (+Magic method)

빌트인 라이브러리 함수에 기능을 추가해보자!

예제8

```
# int 클래스 버전 업데이트 하기 위해 생성한 클래스
class int_v4(int):
    # 인스턴스끼리 덧셈 연산할 때의 함수
    def __add__(self, x):
        return self - x

# int 클래스 버전4의 인스턴스 생성
a = int_v4(3.5)

print(a+4) # 출력: -1
print(4+a) # 출력: 7
```

**Magic Method**

**__add__ : 클래스 덧셈 함수**

# 빌트인 라이브러리 함수 기능 추가하기 (+Magic method)

빌트인 라이브러리 함수에 기능을 추가해보자!

예제8

```
# int 클래스 버전 업데이트 하기 위해 생성한 클래스
class int_v4(int):
    # 인스턴스끼리 덧셈 연산할 때의 함수
    def __add__(self, x):
        return self - x

# int 클래스 버전4의 인스턴스 생성
a = int_v4(3.5)
print(a+4) # 출력: -1
print(4+a) # 출력: 7
```

int_v4의 인스턴스인 self 가 지칭하는 것

**Magic Method**

**__add__ : 클래스 덧셈 함수**

# 빌트인 라이브러리 함수 기능 추가하기 (+Magic method)

빌트인 라이브러리 함수에 기능을 추가해보자!

예제8

```
# int 클래스 버전 업데이트 하기 위해 생성한 클래스
class int_v4(int):
    # 인스턴스끼리 덧셈 연산할 때의 함수
    def __add__(self, x):
        return self - x

# int 클래스 버전4의 인스턴스 생성
a = int_v4(3.5)

print(a+4) # 출력: -1
print(4+a) # 출력: 7
```

**Magic Method**

**__add__ : 클래스 덧셈 함수**

**일반 int 클래스의 인스턴스 (self) 이기 때문에 int_v4의 self가 아님!**

**따라서 int_v4의 오버라이딩된 __add__를 사용할 수 없음!**

**이럴 경우에는 덧셈의 순서가 굉장히 중요함!**

# 빌트인 라이브러리 함수 기능 추가하기 (+Magic method)

빌트인 라이브러리 함수에 기능을 추가해보자!

예제9

```
# int 클래스 버전 업데이트 하기 위해 생성한 클래스
class int_v4(int):
    # 인스턴스끼리 덧셈 연산할 때의 함수
    def __add__(self, x):
        return self - x

# int 클래스 버전4의 인스턴스 생성
a = int_v4(3.5)
b = int_v4(6.5)

print(a+b) # 출력: -3
print(b+a) # 출력: 3
```

**Magic Method**

**__add__ : 클래스 덧셈 함수**

**a와 b변수 모두 int_v4의 인스턴스 이기 때문에 덧셈 순서에 상관없이 오버라이딩된 __add__ 연산 가능**