

# 클래스의 이해 - 2

---

송기태 ([kitae040522@gmail.com](mailto:kitae040522@gmail.com))  
Soongsil Univ. (Computer Science and Engineering)

# Content

클래스 상속

클래스 메소드 오버라이딩

클래스 다중 상속

실습: 텍스트 RPG 게임 만들기

## 상속(inheritance)이란?

- 클래스 상속은 부모 클래스의 속성과 메소드를 자식 클래스가 물려받는 것을 의미
- 자식클래스를 선언할 때 소괄호로 부모클래스를 포함해야 함
- 자식클래스에서는 따로 부모클래스의 속성과 메소드를 기술하지 않아도 자동으로 상속됨
- ex)
  - 부모 클래스: 국가
  - 자식 클래스: 한국, 일본, 중국, 미국 등
  - 국가 클래스에 인구 속성이 있다고 가정한다면, 이 속성은 한국, 일본, 중국 등등의 클래스에서 사용 가능


## 부모 클래스 정의하기

```
class Country:
    def __init__(self, name, population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show(self):
        print("부모 클래스입니다.")
```

## 자식 클래스 정의하기

```
class Korea(Country):  
    def __init__(self, name, population, capital):  
        super().__init__(name, population, capital)  
        self.greeting = "안녕하세요!"  
  
    def greet(self):  
        print(self.greeting)
```



이 키워드를 통해서, 부모 클래스의 생성자를 호출해줘야한다.

## 자식 클래스 정의하기

```
class Japan(Country):  
    def __init__(self, name, population, capital):  
        super().__init__(name, population, capital)  
        self.greeting = "こんにちは!"  
  
    def greet(self):  
        print(self.greeting)
```

## 자식 클래스 정의하기

```
if __name__ == '__main__':  
    def main():  
        korea_obj = Korea("대한민국", 51_740_000, "서울")  
        japan_obj = Japan("일본", 125_700_000, "도쿄")  
        korea_obj.show() # 출력 결과: 부모 클래스입니다.  
        korea_obj.greet() # 출력 결과: 안녕하세요!  
        print(japan_obj.population) # 출력 결과: 125700000  
        japan_obj.greet() # 출력 결과: こんにちは!  
    main()
```

## 메소드 오버라이딩(Method Overriding)이란?

- 자식 클래스가 부모 클래스로부터 상속받은 메소드를 동일한 이름의 메소드로 다시 정의하는 것을 의미
- 자식 클래스에서 부모 클래스의 메소드를 덮어쓰거나 재정의 가능



# 클래스 메소드 오버라이딩

```
class Parent:
    def display(self):
        print("부모입니다.")

class Child(Parent):
    def display(self):
        print("자식입니다.")

if __name__ == '__main__':
    def main():
        app = Child()
        app.display()    # 출력 결과: 자식입니다.

    main()
```

# 클래스 메소드 오버라이딩

```
import datetime

class Logger:
    def log(self, msg):
        print(msg)

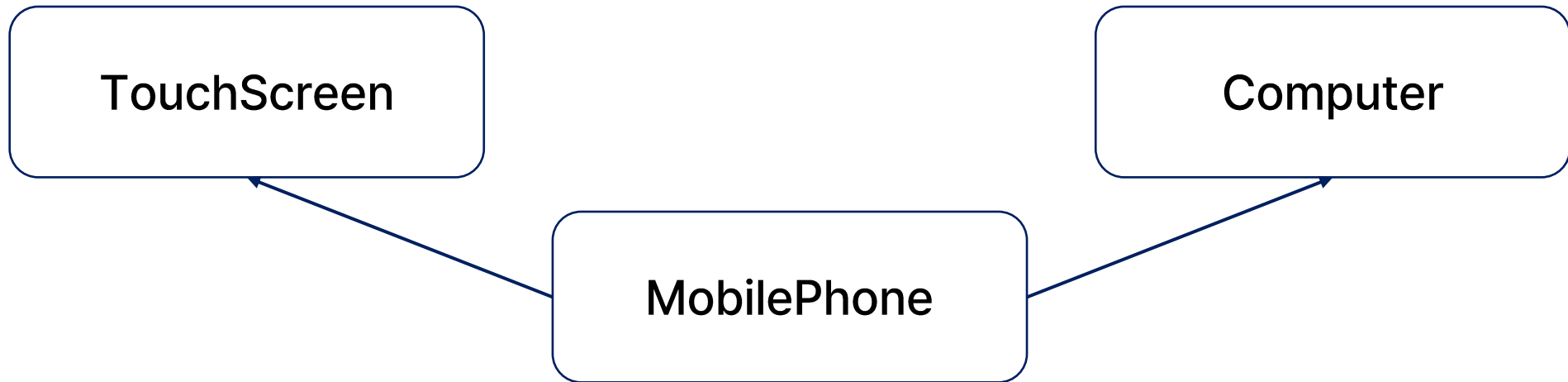
class TimestampLogger(Logger):
    def log(self, whoami):
        msg = f"{whoami} >> {datetime.datetime.now()}"
        super().log(msg)

class DateLogger(Logger):
    def log(self, whoami):
        msg = f"{whoami} >> {datetime.datetime.now().strftime('%Y-%m-%d')}"
        super().log(msg)
```

# 클래스 메소드 오버라이딩

```
if __name__ == '__main__':  
    def main():  
        l = Logger()  
        t = TimestampLogger()  
        d = DateLogger()  
        l.log("Logger") # 출력 결과: Logger  
        t.log("TimestampLogger") # 출력 결과: TimestampLogger >> 2023-09-20 02:07:58  
        d.log("DateLogger") # 출력 결과: DateLogger >> 2023-09-20  
  
    main()
```

### 여러 개의 부모 클래스로부터 상속받는 것을 의미



# 클래스 다중 상속

```
class TouchScreen:
    def touch(self, x, y): pass

class Computer:
    def compute(self): pass

class MobilePhone(TouchScreen, Computer):
    def __init__(self, size):
        self.size = size

    def touch(self, x, y):
        print(f"x축 {x}, y축 {y}를 터치했습니다.")

    def compute(self):
        print("계산 중입니다.")
```

## 클래스 다중 상속

```
if __name__ == '__main__':  
    def main():  
        phone = MobilePhone(6.1)  
        phone.touch(1920, 1080) # 출력 결과: x축 1920, y축 1080를 터치했습니다.  
        phone.compute() # 출력 결과: 계산 중입니다.  
  
    main()
```

## 다이아몬드 상속 문제 (Diamond Problem)

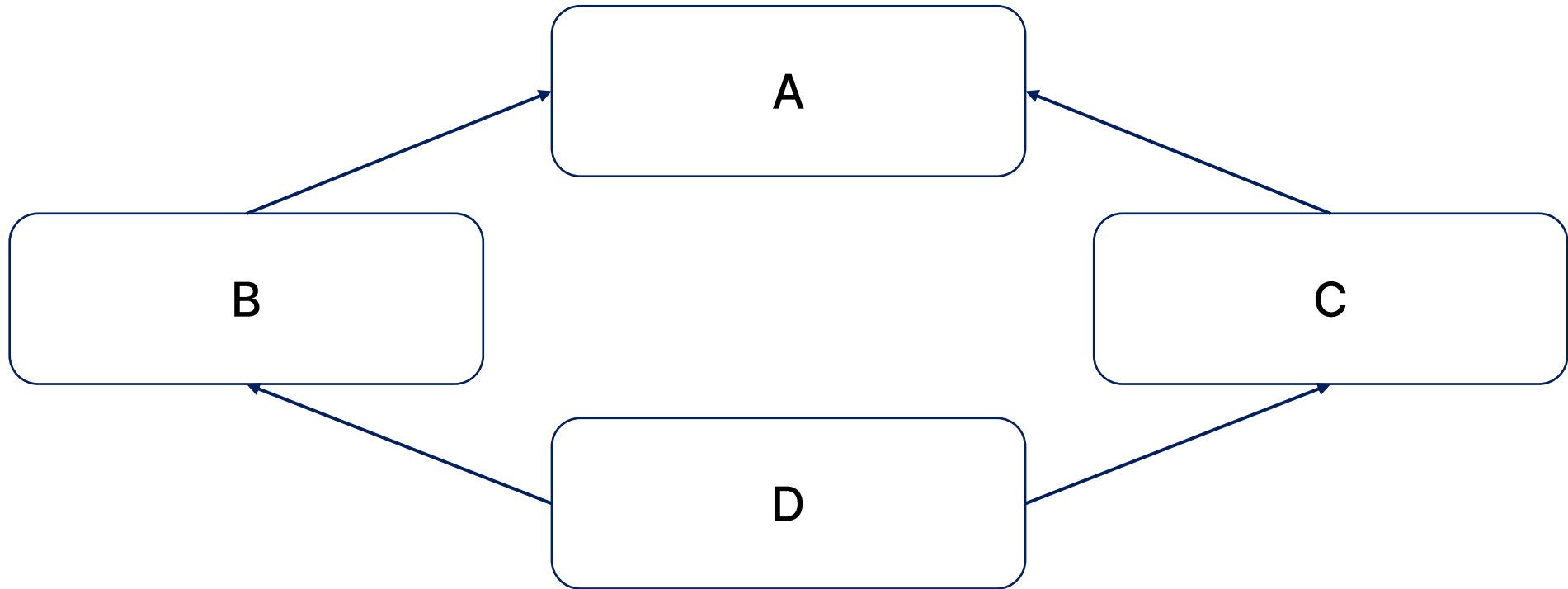
```
class A:
    def introduce(self):
        print("A")

class B(A):
    def introduce(self):
        print("B")

class C(A):
    def introduce(self):
        print("C")

class D(B, C):
    pass
```

## 다이아몬드 상속 문제 (Diamond Problem)





## 다이아몬드 상속 문제 (Diamond Problem)

```
if __name__ == '__main__':  
    def main():  
        x = D()  
        x.introduce()  
  
    main()
```

Q. x.introduce()의 출력결과는 무엇이 될까?

1. "C"
2. "B"

## 다이아몬드 상속 문제 (Diamond Problem)

메소드 결정 순서(MRO, Method Resolution Order)를 사전에 확인하여 혹시 모르는 문제를 예방하자!

```
>>> print(D.mro())  
# [<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```

D의 메소드 결정 순서는 자기 자신(D), B, C, A이다.

D로 인스턴스를 만들고, D에는 introduce 메소드가 존재하지 않기 때문에, B의 introduce 메소드가 호출된다.

# 실습: 텍스트 RPG 게임 만들기

```
import random
import math

class Object:
    def __init__(self, name, hp, power):
        self.name = name
        self.hp = hp
        self.power = power
        self.ability_power = 0

    def attack(self, target_class):
        print(f"{self.name}이 {target_class.name}을 {self.power}만큼 공격했습니다!")
        target_class.hp -= self.power
        if self.is_dead(target_class):
            print(f"{target_class.name}을 물리쳤습니다!")
        else:
            print(f"{target_class.name}을 물리치지 못했습니다.")

    def is_dead(self, target_class):
        if target_class.hp <= 0:
            return True
        return False
```

# 실습: 텍스트 RPG 게임 만들기

```
class Player(Object):
    def __init__(self, name, hp, power, job):
        super().__init__(name, hp, power)
        self.job = job
        self.money = 0
        self.critical_hit_probability = 0.3

class Warrior(Player):
    def __init__(self, name, hp, power, job):
        super().__init__(name, hp, power, job)

    def attack(self, target_class):
        dmg = self.power
        if self.critical_hit_probability > random.random():
            dmg = math.ceil(dmg * 1.5)
            print(f"크리티컬!")
        print(f"{self.name}이 {target_class.name}을 {dmg}만큼 공격했습니다!")
        target_class.hp -= dmg
        if self.is_dead(target_class):
            print(f"{target_class.name}을 물리쳤습니다!")
        else:
            print(f"{target_class.name}을 물리치지 못했습니다.")
```

# 실습: 텍스트 RPG 게임 만들기

```
class Wizard(Player):
    def __init__(self, name, hp, power, job):
        super().__init__(name, hp, power, job)
        self.ability_power = power + 1
        self.power = 0

    def attack(self, target_class):
        print(f"{self.name}이 {target_class.name}을 {self.ability_power}만큼 공격했습니다!")
        target_class.hp -= self.ability_power
        if self.is_dead(target_class):
            print(f"{target_class.name}을 물리쳤습니다!")
        else:
            print(f"{target_class.name}을 물리치지 못했습니다.")

class Monster(Object):
    def __init__(self, name, hp, power):
        super().__init__(name, hp, power)

    def cure(self):
        add_hp = random.randint(1, 5)
        self.hp += add_hp
        print(f"{self.name}이 {add_hp}만큼 회복했습니다!")
```

# 실습: 텍스트 RPG 게임 만들기

```
class TextRPG:
    def __init__(self):
        self.player = None
        self.monster = None

    def print_guide(self):
        print("=" * 30)
        print(f"{self.player.name}의 남은 피: {self.player.hp} / {self.monster.name}의 남은 피: {self.monster.hp}")

    def create_obj(self):
        player_job = int(input("원하는 직업을 선택하세요.\n"
                                "1. 전사\n"
                                "2. 마법사\n"
                                ">> "))

        nickname = input("소환사의 이름을 적어주세요.\n"
                           ">> ")

        if player_job == 1:
            self.player = Warrior(nickname, 10, 5, "전사")
        else:
            self.player = Wizard(nickname, 10, 5, "마법사")
        self.monster = Monster("고블린", 25, 3)
```

# 실습: 텍스트 RPG 게임 만들기

```
def run(self):
    self.create_obj()
    while True:
        self.print_guide()
        self.player.attack(self.monster)
        if self.monster.hp <= 0:
            break

        self.print_guide()
        monster_next_move = random.choice(["attack", "cure"])
        if monster_next_move == "attack":
            self.monster.attack(self.player)
        else:
            self.monster.cure()

        if self.player.hp <= 0:
            break

if __name__ == '__main__':
    def main():
        app = TextRPG()
        app.run()

    main()
```

# Thank You!

---

송기태 ([kitae040522@gmail.com](mailto:kitae040522@gmail.com))  
Soongsil Univ. (Computer Science and Engineering)