지금까지는 일반적인 '은행 계좌'를 뜻하는 BankAccount 클래스로 강의를 진행해왔는데요.

```
public class BankAccount {
    private int balance;
    public int getBalance() {
        return balance;
    }
    public void setBalance(int balance) {
        this.balance = balance;
    }
    // 출금
    public boolean withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
            return true;
        }
        return false
    }
    // 입금
    public boolean deposit(int amount) {
        balance += amount;
        return true;
    }
}
```

실제로는 더 다양한 종류의 은행 계좌가 존재합니다. 예를 들어서 최소 금액을 정해두고 항상 그 이상의 잔액이 있어야 하는 MinimumBalanceAccount 가 있을 수 있고, 잔액에 이자가 붙는 SavingsAccount 도 있을 수 있겠죠?

그런데 잘 생각해보면 BankAccount, MinimumBalanceAccount, SavingsAccount 모두 공통된 속성과 기능이 많이 있습니다. 예를 들어서 모두 잔액 속성과 입금 기능이 있습니다. 만약 이 공통된 부분을 클래스마다 다시 쓴다면 프로그래밍의 중요한 법칙 중 하나인 'DRY(Don't Repeat Yourself; 중복 배체)'를 어기게 되는 것입니다.

다행히 자바의 '클래스 상속(Class Inheritance)' 기능이 이 문제를 해결해줍니다!

## **SavingsAccount**

클래스를 선언할 때 이렇게:

2018. 3. 12. 코드잇

public class SavingsAccount extends BankAccount {

extends BankAccount 를 쓰면, BankAccount 클래스의 속성과 기능을 모두 상속받겠다는 뜻입니다. 그러니까 balance 변수나, deposit 메소드를 다시 정의할 필요가 없는 거죠!

이 경우 BankAccount 는 SavingsAccount 의 '부모 클래스(Parent Class)'라고 부르고, SavingsAccount 는 BankAccount 의 '자식 클래스(Child Class)'라고 부릅니다.

그러면 BankAccount 클래스에는 없는, 추가로 필요한 내용만 써보겠습니다.

```
public class SavingsAccount extends BankAccount {
    private double interest;

    public void setInterest(double interest) {
        this.interest = interest;
    }

    public double getInterest() {
        return interest;
    }

    public void addInterest() {
        setBalance((int) (getBalance() * (1 + interest)));
    }
}
```

### 부모 클래스의 private 변수

여기서 좀 이해가 안 될 수도 있는 부분은 마지막 addInterest 메소드입니다. 왜 그냥 이렇게 쓰지 않 았을까요?

```
balance = (int) (balance * (1 + interest));
```

balance 는 부모 클래스인 BankAccount 의 private 변수이기 때문에 자식 클래스인 SavingsAccount 클래스에서는 사용할 수 없습니다. 따라서 public 메소드인 setBalance 와 getBalance 를 통해 balance 변수에 접근을 하는 것이죠.

#### MinimumBalanceAccount

```
public class MinimumBalanceAccount extends BankAccount {
    private int minimum;
    public void setMinimum(int minimum) {
        this.minimum = minimum;
    }
    public int getMinimum() {
        return minimum;
    }
    @Override
    public boolean withdraw(int amount) {
        if (getBalance() - amount < minimum) {</pre>
            System.out.println("적어도 " + minimum + "원은 남겨야 합니다.");
            return false;
        }
        setBalance(getBalance() - amount);
        return true;
    }
}
```

MinimumBalanceAccount 의 withdraw 메소드는 부모 클래스인 BankAccount 의 withdraw 메소드와 조금 다릅니다. 출금 후 잔액이 minimum 이상이어야 하기 때문이죠.

지금과 같이 자식 클래스가 부모 클래스가 가지고 있는 메소드를 덮어 쓰고 싶을 때는 '메소드 오버라이 딩(Method Overriding)'을 해줘야 합니다. 메소드 정의 위에 써져있는 @Override 가 메소드 오버라이딩 을 표시해줍니다.

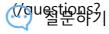
@Override 와 같이 골뱅이(@)가 붙어있는 문법을 '어노테이션(Annotation)'이라고 합니다. 주석 (Comment)과 어느정도 비슷하지만, 어노테이션은 자바에서 추가적인 기능을 제공합니다. 예를 들어서 @Override 를 써줬는데 부모 클래스에 같은 이름의 메소드가 없는 경우, 오류가 나오게 됩니다!

이렇게 하면 MinimumBalanceAccount 의 인스턴스는 BankAccount 의 withdraw 가 아닌 MinimumBalanceAccount 의 withdraw 를 호출하게 됩니다.



✔ 수업을 완료하셨으면 체크해주세요.

♀♀ 수강생 Q&A 보기



#### 코드잇

# assignment\_id=418&sort\_by=popular) (/questions/new?

이전 강의 상속 (/assignments/413) assignment\_id=418&op1=%EA%B0%9D%EC%B2%B4+%EC%A7