21.클래스

클래스

- 클래스 의 정의
  . 구현하고자 하는 대상을 모듈(파트) 별로 나누어 미리 설계(프로그래밍) 하여 두고 로직 상 필요 한 부분에 호출하여 적시적소에 사용 할 수 있게 해둔 프로그램 로직의 모음.
    > Ex) 1. 난수를 생성 하는 기능을 하나의 클래스 에 정의 해 둔다.
    2. 프로그래밍 중 난수 를 생성 하는 기능이 필요 할때 마다 난수 생성 클래스 의 기능을 호출하여 사용한다.
- 클래스 사용의 장점
  . 프로그래밍 중 반복적으로 구현되는 로직이 있을경우 매번 구현 해야 하는 번거로움을 피할 수 있다.
  . 수정 사항이 발생할 경우 클래스 를 호출 모든 로직에 수정 내역을 일괄 적용 할 수 있다.
    * 모든 프로그래밍 언어는 개발자의 프로그램 생산성 향상 과 편의 성을 위하여 발전 된다.
  . 다른 개발자가 유용한 클래스를 사용 할 수 있도록 배포 가 용이 하다.

- 파이썬과 객체지향
  . 파이썬은 객체지향 언어로 만들어 지지 않았지만 객체 지향 프로그래밍을 훌륭하게 소화 해 내고 있다.
  다만 진짜 객체 지향 언어 (JAVA, C#) 보다는 형식성이 떨어지며 기능도 부족한 편이다.
  . 객체지향 언어를 배우기 위한언어로는 적합하지 않다.

캡슐화

- 연관된 로직의 묶음
  . 클래스 를 통하여 연관된 로직을 하나로 모아 둔 형태 를 캡슐화 라고 한다. (Python 의 캡슐화)

Chap01_intro > 20.Class[클래스].py > ...

```
1  class Account:
2      def __init__(self,balance,bankname) :
3          self.balance = balance
4          self.BankName = bankname
5
6      def deposit(self,money):
7          self.balance += money
8
9      def inquire(self) :
10         print('%s 잔액 은 %d 원 입니다.' % (self.BankName, self.balance))
11
12 sinhan = Account(8000,'신한은행')
13 sinhan.deposit(1000)
14 sinhan.inquire()
15
16 nonghyup = Account(120000,'농협')
17 nonghyup.inquire()
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.exe d:/Python/Chap01_intro/20.Class[클래스].py
신한은행 잔액 은 9000 원 입니다.
농협 잔액 은 120000 원 입니다.
```

>1 ~ 10 행
. 은행업무 기능을 모아둔 클래스 로 캡슐화 한 상태

> 2행 : 클래스를 생성 할때 호출 되는 메서드
> 3행 : 인스턴스 변수 클래스 객체 소멸시 같이 소멸변수
> 6행 : 계좌에 입금 할때 호출하는 메서드

>9행 : 계좌 를 조회 할때 호출하는 메서드

> 12행 : sinhan 이라는 이름 의 Account 클래스 객체 생성
> 16행 : nonghyup 이라는 Account 클래스 객체 생성

클래스 의 객체화

- 클래스 를 사용함을 선언 하고 선언한 이름으로 프로그래밍 할수 있는 상태
  . 클래스 의 작성
    > 은행의 업무 와 관련된 기능을 모아 둔 모듈을 만들기.

```
1  class Account:
2      def __init__(self,balance,bankname) :
3          self.balance = balance
4          self.BankName = bankname
5
6      def deposit(self,money):
7          self.balance += money
8
9      def inquire(self) :
10         print('%s 잔액 은 %d 원 입니다.' % (self.BankName, self.balance))
```

> Account 클래스 의 구성

1) 생성자 멤버

```
2       def __init__(self,balance,bankname) :
3           self.balance = balance
4           self.BankName = bankname
```

__init__ : Account 를 사용하기 위하여 메모리에 등록 할 때 사용되는 변수들에 초기 값을 할당
Account 객체 의 클래스 변수 의 초기화 를 하는곳으로 클래스 사용을 선언(인스턴스 화) 할때 자동으로 호출 되는
특수형태의 메서드
* 두개의 밑줄의 의미 __init__() (언더스코어(_), "매직 메서드" 혹은 "던더 메서드")
파이선에서 사용되는 특수 형태의 메서드 이며 용도 와 이름이 미리 정해 져 있다.

self : Account 클래스 를 사용하기 위해 선언하는 이름 . 객체 라고 한다.
객체를 지칭하는 변수 키워드로서 이름은 고정으로 사용된다.

balance : Account 클래스 를 인스턴스화 한 객체 가 초기 값으로 설정한 첫번째 인수를 받는 변수 (인스턴스 변수)
객체에서 전달하는 값을 받는 변수 로 이름은 임의로 만들 수 있다.

bankname : Account 클래스 를 인스턴스화 한 객체 가 초기 값으로 설정한 두번째 인수를 받는 변수 (인스턴스 변수)
객체에서 전달하는 값을 받는 변수 로 이름은 임의로 만들 수 있다.

* 인스턴스 변수
인스턴스 화 된 객체에서 단독으로 사용되는 변수

```
187 class Car:
188     def __init__(self,year,type) :
189         self.madeyear = year
190         self.type = type
191 
192 car1 = Car(2023,"MAYBACH")
193 car2 = Car(2023,"X5")
194 
195 print(car1.madeyear, car1.type)  # 2023,"MAYBACH"
196 print(car2.madeyear, car2.type)  # 2023,"X5"
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

2023 MAYBACH
2023 X5
```

2) 메서드 멤버

```
6       def deposit(self,money):
7           self.balance += money
```

deposit(self,money) : money 변수 에 전달 받은 값을 받아 클래스 객체 self 의 변수 balance 에 누적한다.
self : deposit() 메서드 를 포함하는 객체. 생성자 생성 시 balance 와 BankName 이라는 클래스 변수 를 가지고 있다.
self.balance += money : deposit() 메서드 가 전달받은 money 인자 의 값을 객체 변수 balance 에 누적 한다.

```
9       def inquire(self) :
10          print('%s 잔액 은 %d 원 입니다.' % (self.BankName, self.balance))
```

inquire(self) : inquire() 메서드를 호출하는 객체 를 self 변수 로 전달한다.
print : self 가 가지고 있는 balance 변수 와 BankName 변수 의 값을 호출하여 출력한다.

. 클래스 의 인스턴스 화 및 활용

> Account 클래스 를 사용 하기 위하여 선언 하고 상황에 맞게 사용하기

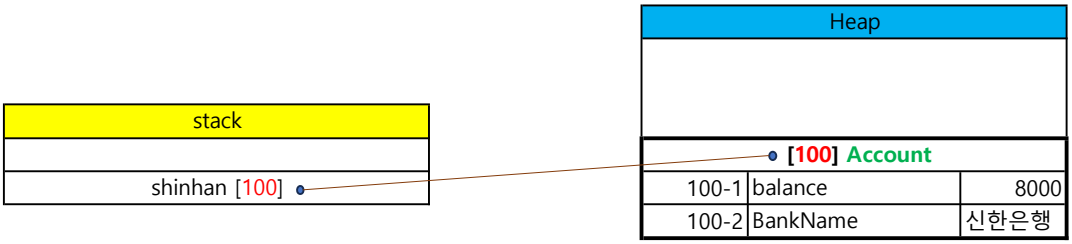
```
12  sinhan = Account(8000,'신한은행')
13  sinhan.deposit(1000)
14  sinhan.inquire()
15
16  nonghyup = Account(120000,'농협')
17  nonghyup.inquire()
```

1) Account 클래스 의 인스턴스 화 (=객체화)

* 인스턴스 : 1회성으로 사용하고 끝이 날 어떤것

```
12  sinhan = Account(8000,'신한은행')
```

Account(8000,'신한은행') : Account 클래스 를 메모리에 등록 한다. (인스턴스 화)
이때 8000 값과 '신한은행' 이라는 초기값을 전달한다.
sinhan = : 인스턴스 화 한 Account 클래스 를 sinhan 이라는 객체 이름에 주소를 전달한다.

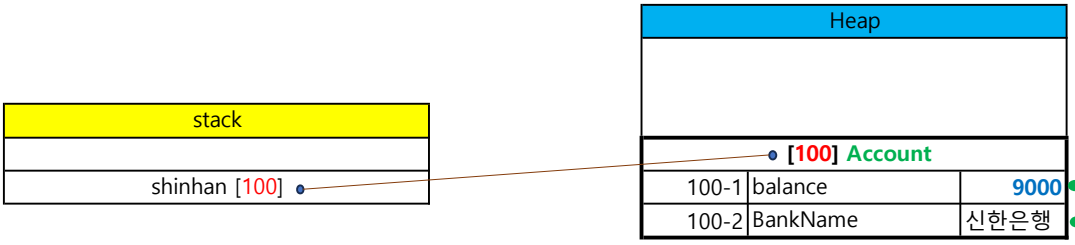


2) Account 클래스 객체 shinhan 의 deposit() 메서드 호출

```
13  sinhan.deposit(1000)
```

```
def deposit(self,money):
    self.balance += money
```

. balance 변수 에 1000 이 누적 되어 9000 으로 변경



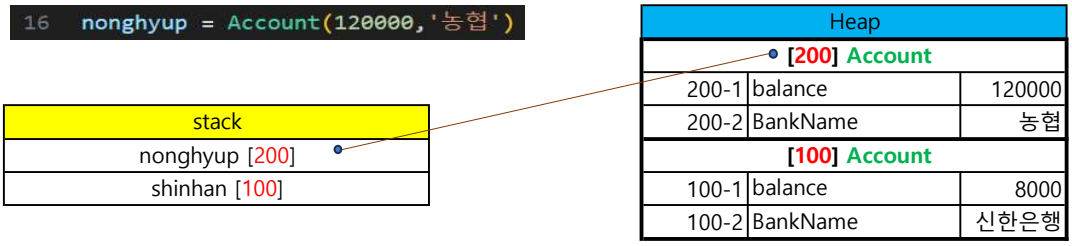
3) shinhan 객체 의 inquire 메서드 호출

```
14  sinhan.inquire()
```

```
def inquire(self) :
    print('%s 잔액 은 %d 원 입니다.' % (self.BankName, self.balance))
```

4) Account 클래스 의 인스턴스 화

```
16  nonghyup = Account(120000,'농협')
```



* Account 클래스 하나 로 전혀 다른 은행의 계좌를 2개 생성 하였다. (클래스의 객체 화 . = 객체 지향 언어)
* sinhan 객체 와 nonghyup 객체 는 서로 다른 Heap 영역 의 Account 클래스 를 참조 하므로 상호 무관한 객체로 사용할 수 있다.

5) 결과 출력

```
신한은행 잔액 은 9000 원 입니다.
농협 잔액 은 120000 원 입니다.
```

상속

- 공통 기능을 물려 주기
  . 비슷한 기능을 하는 클래스 를 n 개 생성 해야 할 경우 공통 적인 부분이 있을 수 있다.
  . 공통 기능을 구현하는 클래스를 구현해 두고(부모 클래스) 부모 클래스를 상속 받는 자식 클래스는 각각 특징적인 기능을 구현 한다.
  * 공통 기능 을 하나로 만들어 두고 호출 해 사용하여 재 사용성과 유지 보수 성을 향상 시킬 수 있다.
  . 상속의 예제 소스

Chap01_intro > 20.Class[클래스].py > Student > intro

```
1  class Human :
2      def __init__(self, age,name):
3          self.age = age
4          self.name = name
5      def intro(self) :
6          print('저는 %d 살 %s 입니다.' % (self.age,self.name))
7
8  class Student(Human):
9      def __init__(self,age,name,stunum):
10         super().__init__(age,name)
11         self.stuname = stunum
12
13     def intro(self):
14         super().intro()
15         print('학번 : ' + str(self.stuname))
16
17 kim = Human(36,'김범수')
18 kim.intro()
19 lee = Student(39,'이수',1120129)
20 lee.intro()
```

>1행 : Human 클래스 작성
>2행 : Human 클래스 생성자 호출 및 인스턴스변수 초기화
>5행 : Human 클래스 의 메서드 작성

>8행 : Student 클래스 생성
* Human 클래스를 상속 받는다.
>9행 : Student 클래스 생성자 호출
>10행: 부모 클래스 인 Human 클래스 의 생성자 호출
>11행: Student 클래스 의 인스턴스 변수 초기화

>13행 : Student 클래스의 intro 메서드 작성
>14행 : 부모 클래스 intro 메서드 실행
>15행 : 자식 클래스 만의 기능 실행

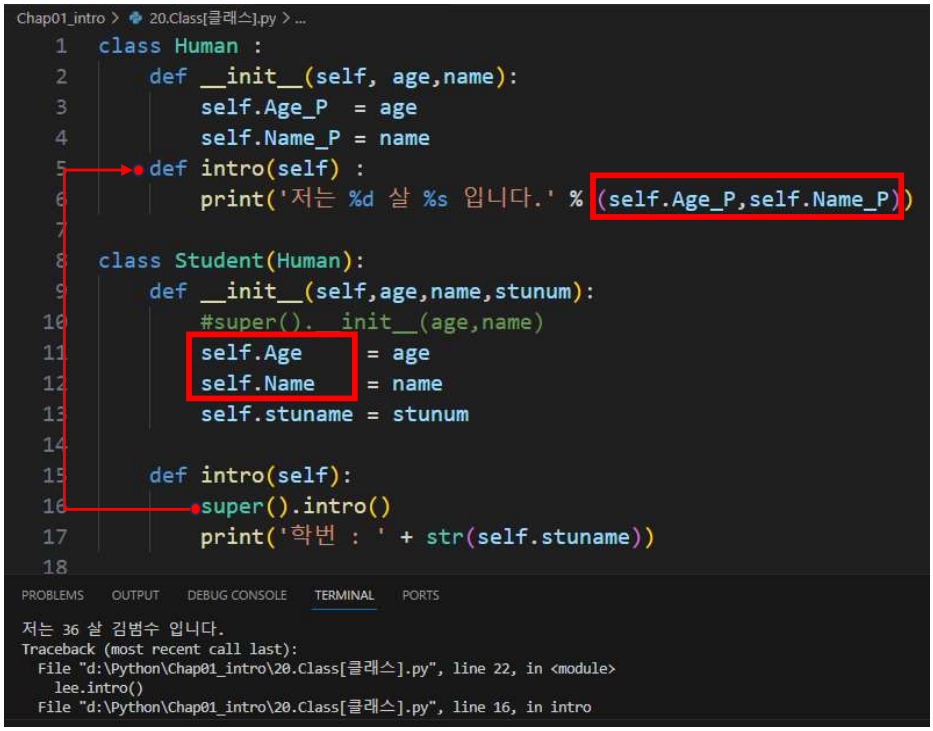
> 실행 결과

```
D:\Python>C:/Users/MasterD/AppData
저는 36 살 김범수 입니다.
저는 39 살 이수 입니다.
학번 : 1120129
```

* super() : Student 클래스에 기능을 상속하여 준 부모 클래스 의 기능을 호출 한다.
lee 클래스 의 intro 클래스 를 호출 하였을 경우 부모 클래스의 intro() 를 실행 후 학번을 소개하는 출력 문이 동작 되는것을 볼 수있다.
* 부모 클래스 의 __init__()
. 부모 클래스에서 물려 준 인스턴스 변수 를 그대로 사용할 경우 부모 클래스의 생성자 를 호출 하여 실행 한다.
. 자식 클래스 에서 도 초기 화 할 수 있지만. 부모 클래스의 수정이 일어날 경우 자식 클래스도 수정해야 하는 비효율성이 발생한다.

```
class Student(Human):
    def __init__(self,age,name,stunum):
        #super().__init__(age,name)
        self.age     = age
        self.name    = name
        self.stuname = stunum
```

> 부모 클래스 가 물려준 인스턴스 변수 를 자식 클래스 에서 초기화 할 수 있다 .
* self.age , 와 self.name 는 부모가 물려준 인스턴스 변수 를 지칭한다.



```
Chap01_intro > 20.Class[클래스].py > ...
1  class Human :
2      def __init__(self, age,name):
3          self.Age_P  = age
4          self.Name_P = name
5      def intro(self) :
6          print('저는 %d 살 %s 입니다.' % (self.Age_P,self.Name_P))
7
8  class Student(Human):
9      def __init__(self,age,name,stunum):
10         #super().__init__(age,name)
11         self.Age    = age
12         self.Name   = name
13         self.stuname = stunum
14
15     def intro(self):
16         super().intro()
17         print('학번 : ' + str(self.stuname))
18

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

저는 36 살 김범수 입니다.
Traceback (most recent call last):
  File "d:\Python\Chap01_intro\20.Class[클래스].py", line 22, in <module>
    lee.intro()
  File "d:\Python\Chap01_intro\20.Class[클래스].py", line 16, in intro
```

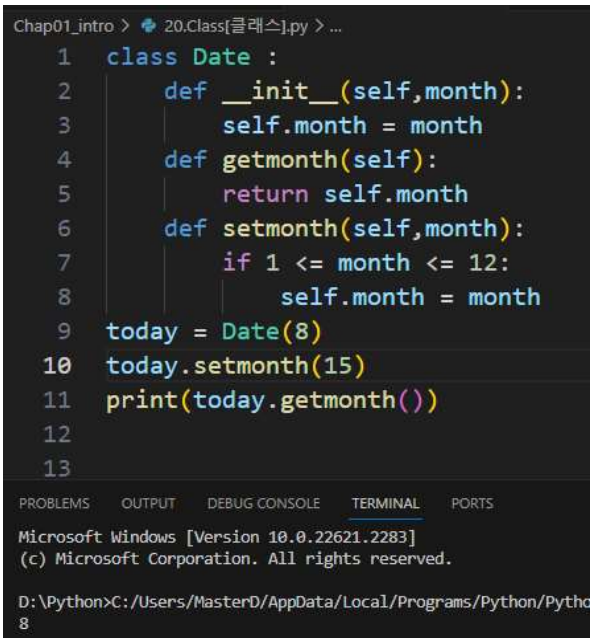
> Student 클래스만의 인스턴스 변수 Age 와 Name 가 생성된다.

> 부모 클래스의 인스턴스 변수 명을 변경 하였지만 자식 클래스에서 부모 클래스의 인스턴스 변수 를 초기화 하지 않으면 오류 가 발생한다.
* 자식 클래스와 부모 클래스 간 관계없는 Age_P 와 Age 인스턴스 변수 가 생성 되었기 때문에 자식 클래스 에서 부모 클래스의 intro() 호출(16행) 시 부모 클래스의 Age_P 는 초기화 (값이 할당된 상태) 가 아니므로 오류가 발생하는것

접근 제한

- 파이썬 클래스의 멤버는 모두 공개 되어 있으며 외부에서 누구 나 엑세스 할 수 있다.
  . 접근 제한의 문법이 존재 하지만 사실상 객체지향 언어의 기능을 모두 구현하는것은 어렵다

- 클래스 데이터 접근을 막아 변질을 방지하는 제한 방법
  . 일정한 규칙을 마련해 두고 외부의 요인으로부터의 변질을 막기
  . 1 ~ 12 사이의 값만 처리 할수 있도록 하는 month 인스턴스 변수 의 변질을 막는 방법



```
Chap01_intro > 20.Class[클래스].py > ...
1  class Date :
2      def __init__(self,month):
3          self.month = month
4      def getmonth(self):
5          return self.month
6      def setmonth(self,month):
7          if 1 <= month <= 12:
8              self.month = month
9  today = Date(8)
10 today.setmonth(15)
11 print(today.getmonth())
12
13

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Pytho
8
```

>10행 : setmonth 메서드 에 15를 전달한다.
>7행 : 전달 받은 month 정보 가 1~12 일경우만 month 변수 에등록한다.
>결과 : 15 는 범위에 속하지 않으므로 초기값 8 이 출력된다.

. Property 클래스 를 이용한 데이터 변질 막기

> Property : 특정 데이터를 읽을때 조건을 줄 수 있는 메서드 와 특정 데이터 를 갱신할 때 조건을 주는 메서드 를 인자로 받아 외부에서 특정 데이터의 변질을 막을 수 있다.

```
Chap01_intro > 20.Class[클래스].py > ...
1  class Date :
2      def __init__(self,month):
3          self.inner_month = month
4      def getmonth(self):
5          return self.inner_month
6      def setmonth(self,month):
7          if 1 <= month <= 12:
8              self.inner_month = month
9      month = property(getmonth,setmonth)
10
11 today = Date(8)
12 today.month = 15
13 print(today.month)
14

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python
8
```

>12행 : today 객체의 month 프로퍼티에게 15를 전달
>9행 : 15를 전달 받은 month 프로퍼티가 setmonth 구동
>7행 : 1~12 사이 숫자인지비교 후 인스턴스 변수 inner_month 에 값 대입

. 데커레이터 를 통한 프로퍼티의 정의

> 위 예제의 9행 (프로퍼티 생성) 대신 4행 과 8 행 의 데커레이터 가 추가 되었다.

* 데코레이터 : 파이썬의 함수나 메서드의 기능을 확장하거나 수정하는 기능을 한다. 메서드 를 위한 메서드 (어노테이션 이라고 표현 가능 @ 기호는 '엣' 이라고 읽는다)

```
Chap01_intro > 20.Class[클래스].py > ...
1  class Date :
2      def __init__(self,month):
3          self.inner_month = month
4      @property
5      def month(self):
6          return self.inner_month
7
8      @month.setter
9      def month(self,month):
10         if 1 <= month <= 12:
11             self.inner_month = month
12
13 today = Date(8)
14 today.month = 15
15 print(today.month)
16

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/py
8
```

>4행 : 5행의 month 메서드 를 month 의 Getter 로 사용

>8행 : 9행의 Month 메서드 를 month 의 Setter 로 사용