Python Fundamental

Ch.11 : Pemrograman Beorientasi Objek

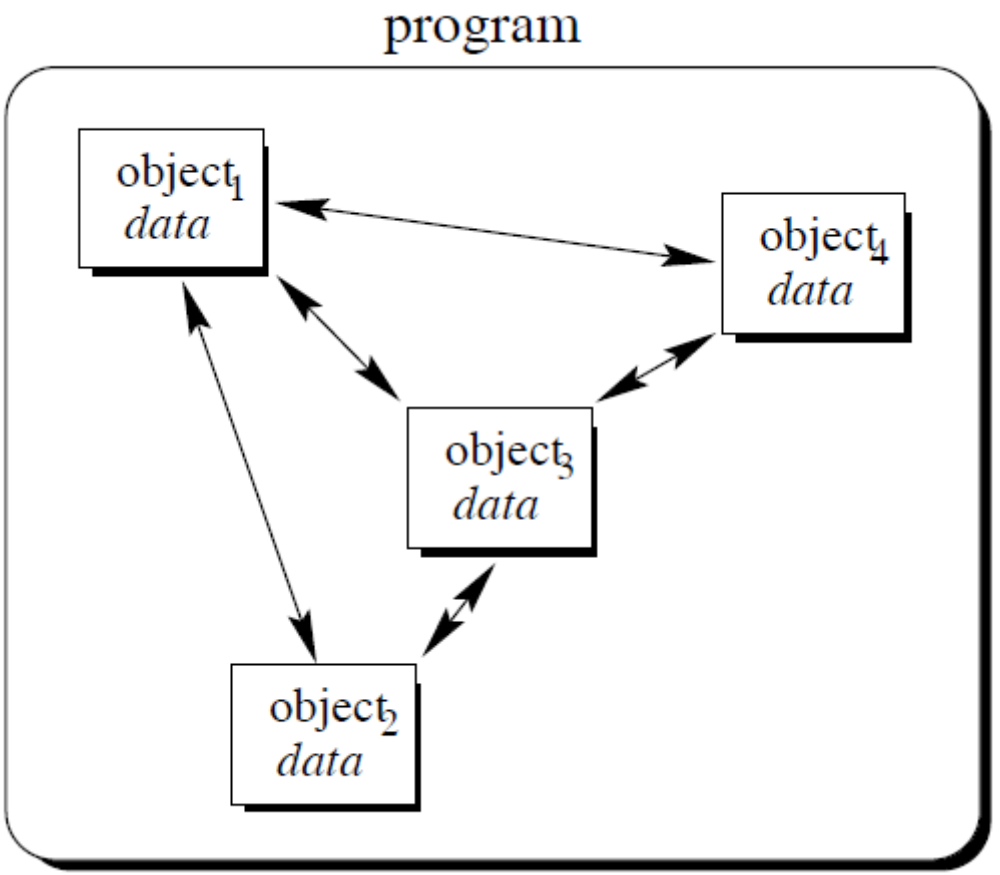
Instruktur : Ahmad Rio Adriansyah
Nurul Fikri Komputer

Instructor

- Ahmad Rio Adriansyah
- 081573954126
- ahmad.rio.adriansyah@gmail.com
- arasy.net

OOP

- Object Oriented Programming
- Object = bagian software yang memiliki variabel dan fungsi
- Programnya didesain meniru benda dan interaksinya di dunia nyata



OOP

- Fokus dari OOP adalah :
  - **Encapsulation**
    Penyimpanan variabel dan fungsi yang berkaitan dalam satu tempat, dan izin aksesnya diatur
  - **Polymorphism**
    Penanganan operator yang memiliki nama sama, tetapi memiliki perilaku yang berbeda tergantung dengan konteksnya
  - **Inheritance**
    Pewarisan variabel dan fungsi dari suatu kelas kepada kelas lain dengan memungkinkan penambahan lainnya

- Objek kucing:
  - Memiliki variabel
    - Warna
    - Ras
    - Panjang ekor, dll
  - Memiliki fungsi
    - Berjalan
    - Bersuara
    - Makan

Class

- Kelas adalah definisi dari objek yang nantinya akan dibentuk
- Analoginya sama seperti blueprint sebuah gedung.
  - Kelas = blueprint
  - Objek = gedung

- Objek adalah benda yang dibentuk dari kelas (disebut sebagai ***instance***)

Class

- Format :

  >>> class nama_kelas :

  … def fungsi():

  … expression

  … variabel

- Instansiasi :

  >>> nama_instance = nama_kelas()

Contoh

```
>>> class kucing:
...         def bersuara(self):
...                 print('Meooong')
...         warna = 'biru'
>>>
>>> Odi = kucing()
>>> Odi.bersuara()
>>> Odi.warna
```

Constructor

- Konstruktor adalah fungsi yang dipanggil pada saat objek di instansiasi
- Format :

```
...     def __init__(self, var):
...             expression
```

Contoh

```
>>> class kucing:
...         def __init__(self, nama= '', warna = '', ras= 'kampung')
...             self.nama = nama
...             self.ras = ras
...             self.warna = warna
...             print('Telah dibuat seekor kucing dengan nama %s'
%nama)
...         def bersuara(self):
...             print('Meoooong')
```

Print Representation

- Kita dapat mengganti bagaimana python menerapkan fungsi print terhadap objek

```
...     def __repr__(self):
...         return 'Halo, saya kucing %s berwarna %s. Nama saya %s' %(self.ras, self.warna, self.nama)
```

Encapsulation

- Semua data pada latihan sebelumnya bersifat publik, artinya dapat diakses dari manapun termasuk diubah dari luar kelasnya
- Contoh :

  >>> Odi.ras = ‘anggora’

- Pada objek, kadang kita perlu suatu nilai yang tidak dapat diubah dengan mudah. Kita lindungi nilai tersebut dengan ***encapsulation (data hiding)*** dan dibuat fungsi setter dan getter
- Enkapsulasi penting, terutama jika ada orang lain yang menggunakan kelas tersebut

Private dan Public Data

- Pada python, variabel yang dimulai dengan **dua buah underscore** (__var) adalah **private**
- Variabel yang dimulai dengan **sebuah underscore** adalah **semi private**
- Setter = fungsi yang digunakan untuk mengubah data private
- Getter = fungsi yang digunakan untuk mengambil nilai data private

Contoh

```
…    def __init__(self, nama='', warna='', ras= 'kampung'):
…        self.nama = nama
…        self.__warna = warna   #private
…        self.__ras = ras   #private
…    def set_warna(self,warna):   #setter
…        self.__warna = warna
…    def get_warna(self):   #getter
…        return self.__warna
…    def get_ras(self):   #getter
…        return self.__ras
```

Encapsulation

- Kenapa dilakukan?
  - Dengan menentukan interface tertentu, kita dapat membatasi modul lain agar tidak melakukan hal yang tidak seharusnya ke data yang ada di kelas kita
  - Membuat kode lebih modular karena kita dapat mengubah kode tanpa mempengaruhi bagian lain dari program (selama bagian lain tersebut hanya mengakses fungsi yang publik)
  - Mirip seperti API (*Application Programming Interface*). Yang lain cukup tahu bagaimana menggunakan bermacam metode pada kelas tersebut, tetapi tidak perlu tahu bagaimana metode itu diimplementasikan

Latihan

- Buat kelas untuk unggas : ayam, bebek, burung
- Ketiganya punya fungsi : bersuara, makan
- Ketiganya punya variabel : kaki, paruh,
- Ayam memiliki fungsi berkokok, kaki = bertaji
- Bebek memiliki fungsi berenang, kaki = berselaput
- Burung memiliki fungsi terbang, kaki = bercakar
- Buat instantiasi masing-masing kelas

Polymorphism

- Ketiga kelas unggas yang dibuat memiliki suara yang berbeda beda, tetapi dipanggil dengan cara yang sama (melalui fungsi bersuara())
- Ini adalah polimorfisme
- Saat ada beberapa kelas yang memiliki fungsi dengan nama yang sama, tetapi memiliki implementasi yang berbeda-beda, maka kelas kelas tersebut polimorphic

Contoh Lain Polymorphism

- Fungsi luas untuk kelas bangun datar yang berbeda-beda (lingkaran, persegi, segitiga)
- Fungsi onClick pada button (cancel, ok, save)
- Fungsi print
- Fungsi operator + terhadap string, int, dan list
- Fungsi operator * terhadap string dan int

Inheritance

- Inheritance adalah situasi dimana sebuah kelas dibangun menggunakan kode dari kelas lainnya
- Analoginya seperti anak yang mewarisi fitur-fitur dari orang tuanya
  - Warna mata, bentuk rambut, atau tinggi badannya diwarisi dari orang tuanya
  - Beberapa budaya anak juga mewarisi nama belakang (marga) dari orang tuanya

Inheritance

- Ada kelas dan sub-kelas dalam dunia nyata
- Contohnya :
  - unggas : ayam, bebek, burung
  - bangun datar : segitiga, lingkaran, persegi
  - pekerja : manajer, teknisi, administrasi
  - tempat tinggal : rumah, apartemen, kos-kosan
  - knight : dark knight, magic knight, guardian
- Kita bisa mewariskan fitur dari kelas ke sub kelasnya

Inheritance

- Yang mewariskan fitur (variabel atau fungsi) disebut sebagai superclassnya
- Contoh :
  - Pekerja : punya NIK, nama, tanggal lahir, alamat ; bisa mengajukan cuti, menerima gaji, dll
  - Manajer : punya semua fitur dari pekerja, tetapi ada tambahan divisi penempatan, bawahan ; bisa mengajukan staf, memberi penilaian, dll
- Kelas manajer adalah sub kelas dari kelas pekerja
- Kelas pekerja adalah super kelas dari kelas manajer

Inheritance

- Format :

```
>>> #parent class
>>> class Parent:
...         expression
>>> #child class
>>> class Child(Parent):
...         expression
```

Contoh

```
>>> class Unggas: #parent class
...         def __init__(self, nama):
...             self.__nama = nama
...             self.kaki = 2
...             print('Sebuah instance dari Unggas telah dibuat')
...         def berbulu(self):
...             print('Unggas berbulu')
...         def berparuh(self):
...             print('Unggas berparuh')
...         def bersuara(self):
...             print('Unggas bersuara')
```

Contoh

```
>>> class Ayam(Unggas): #child class
...         def berkokok(self):
...             print('Ayam jantan berkokok')
...         def bersuara(self): #overriding
...             print('Suara ayam petok petok')
>>> class Bebek(Unggas): #child class
...         def berenang(self):
...             print('Bebek dapat berenang')
...         def bersuara(self): #overriding
...             print('Suara bebek kwek kwek')
```

Contoh

```
>>> kakin = Unggas('kakin')
>>> niwatori = Ayam('niwatori')
>>> ahiru = Bebek('ahiru')
>>> gabung = [kakin, niwatori, ahiru]
>>> for x in gabung:
...         x.bersuara()
>>> niwatori.berkokok()
>>> ahiru.berenang()
```

Tambahan

- Fungsi berikut akan membuat kelas dapat diperlakukan seperti objek lain pada python :

```
__getitem__(self,index) # list
__call__(self,args) # function
__setitem__(self,index,value) # dictionary
__add__(self,other) # operator +
__mul__(self,number) # operator *
```