Algoritma Pemrograman

ABSTRACT DATA TYPE (ADT) MENGGUNAKAN PEMROGRAMAN BERORIENTASI OBJEK (PBO)

NISA'UL HAFIDHOH, M.T.

TIM ALPRO – TI S1

Review

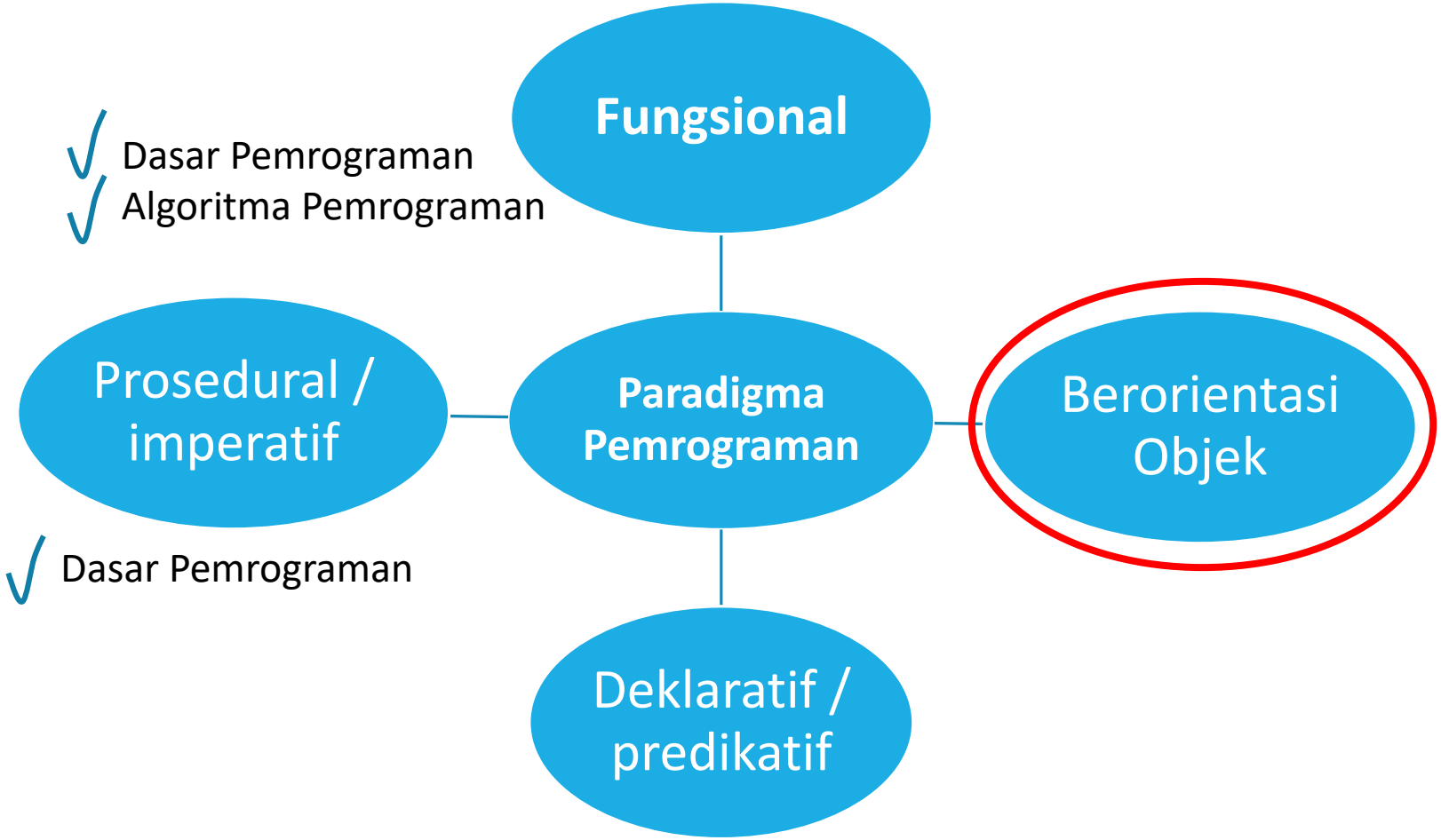
- Sebelumnya sudah memahami dan membuat Abstract Data Type, Objek, dll

Tujuan Pembelajaran

Mahasiswa mampu mengidentifikasi dan menerapkan komponen data dan perilaku dari suatu Abstract data Type (ADT) dan dapat digunakan bersama-sama untuk pemecahan suatu masalah..

Mahasiswa mampu membuat Abstract data Type (ADT) menggunakan Pemrograman Berorientasi Objek (PBO)

Berbagai Paradigma Pemrograman



Abstract Data Type (ADT)

ADT / Tipe data abstrak merupakan jenis spesial dari tipe data yang didefinisikan oleh set dari nilai-nilai dan operasi-operasi

ADT dibentuk oleh tipe data primitif yang sudah ada, tetapi operasi logika yang ada didalamnya disembunyikan.

ADT dicapai dengan Pemrograman Berorientasi Object (di Python)

Pemrograman Berorientasi Objek (PBO)

Pemrograman Berorientasi Objek adalah paradigma pemograman yang menyelesaikan masalah dengan menyediakan objek-objek(terdiri dari beberapa *attribute* dan *method)* yang saling berkaitan

Semua yang ada di python adalah objek

Pemrograman Berorientasi Objek (PBO)

Dapat menciptakan objek baru dengan beberapa tipe data (Konstruktor)

Dapat memanipulasi objek (Manipulator)

Dapat menghancurkan objek (Destruktor) - dimana ketika objek tadi sudah tidak dibutuhkan lagi

- Dilakukan dengan perintah “del” di python
- Sistem python akan mengakui kembali objek yang dihancurkan atau objek yang tidak dapat di akses yang dikenal dengan istilah “garbage collection”

Beberapa Konsep Utama PBO

Abstraction → Abstraksi objek sejenis dengan sifat yang berbeda-beda

Encapsulation → Penyembunyian Informasi

Inheritance → Pewarisan suatu objek

Polimorphism → Suatu objek yang bisa memiliki banyak bentuk

Jadi apa keuntungan PBO?

Mudah menerjemahkan model bisnis ke model implementasi software.

Kemampuan maintain yang efisien dan cepat (karena sudah dibundle dalam suatu paket).

Efektif dalam kerja tim.

Kode dapat dengan mudah beradaptasi dengan third-party code/program.

Mendukung komputasi modern.

Mendukung OS modern.

Konstruktor

Metode untuk menginisialisasi pembuatan instance objek dari kelas tersebut.

Menggunakan fungsi / method **__init__** dan parameter dengan arfumen pertama selalu **self**

```
class Koordinat(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y
```

Manipulator

Metode untuk memanipulasi objek dari kelas tersebut

- Dapat mengakses atribut objek : menggunakan operator titik (.) → **namaClass.namaAtribut**
- Dapat menambah, mengubah, menghapus atribut objek

```
Koor1 = Koordinat(1,1)
Koor1.x = 2
Koor1.y = 3
```

- Cara lainnya menggunakan fungsi yang sudah ada seperti

  getattr(namaObjek, namaAtribut) : mengakses atribut objek

  hasattr(namaObjek, namaAtribut) : mengecek apakah objek memiliki atribut tertentu atau tidak

  setattr(namaObjek, namaAtribut, nilai) : mengubah nilai atribut**,** jika atribut tidak ada, maka atribut tersebut akan dibuatkan

Destruktor

Python dapat menghancurkan objek instance secara otomatis dan bahkan membebaskan memory sepenuhnya (free memory).

Suatu proses dimana python secara periodik mengakui kembali block memory yang tidak lagi digunakan disebut dengan istilah Garbaga Collection.

Gunakan method del

- K = Koordinat(1,2)
- del(K)

Kita juga dapat menggunakan mekanisme destruktor, (kebalikan dari konstruktor) dengan mendefinisikan method __del__(self) didalam suatu class.

```
class Koordinat(object):
  def __init__(self,x,y):
    self.x = x
    self.y = y
  def __str__(self):
    return "<"+str(self.x)+","+str(self.y)+">"
  def __del__(self):
    class_name = self.__class__.__name__
    print(class_name, "destroyed")
```

Built-in class Attributes

Setiap class di python akan secara otomatis memiliki attribut-attribut yang memang sudah disiapkan tanpa harus mendefinisikan atribut tersebut didalam class.

Atribut-atribut built-in ini dapat diakses dengan titik.

- **__dict__** − Dictionary yang berisi namespace dari class.
- **__doc__** − Docstring class.
- **__name__** − nama class.
- **__module__** − Nama modul dimana class tersebut diciptakan. Biasanya mengembalikan main"__main__" pada mode interaktif.
- **__bases__** − Bisa jadi tuple kosong, mengembalikan parent/superclass/base class.

Panggil itu dengan cara:

- print(Koordinat.__dict__)
- print(Koordinat.__doc__)

Kode Lengkap

```
class Koordinat(object):
  def __init__(self,x,y):
    self.x = x
    self.y = y
  def jarak(self,lain):
    x_jar = (self.x - lain.x)**2
    y_jar = (self.y - lain.y)**2
    return (x_jar+y_jar)**0.5
  def __str__(self):
    return "<"+str(self.x)+","+str(self.y)+">"
  def __neg__(self):
   return Koordinat(-self.x,-self.y)
  def __del__(self):
    class_name = self.__class__.__name__
    print(class_name, "destroyed")
```

```
def main():
  titik_c = Koordinat(4,5)
  titik_asal = Koordinat(0,0)
  print("titik c: x=",titik_c.x,"y=",titik_c.y)
  print("titik asal: x=",titik_asal.x,"y=",titik_asal.y)
  titik_c = -titik_c
  print("titik c: x=",titik_c.x,"y=",titik_c.y)
  titik_c.x = 12
  print("titik c: x=",titik_c.x,"y=",titik_c.y)
  print(titik_c)
  print(type(titik_c))
  print(isinstance(titik_c,Koordinat))
  print(titik_c.__dict__)
  print(titik_c.__doc__)
  print(titik_c.__module__)
  print(Koordinat.__name__)
  print(Koordinat.__base__)
  print(Koordinat.__dict__)
  print(Koordinat.__doc__)
  print(Koordinat.__module__)
  del(titik_c)
  del(titik_asal)
  print(titik_c.__dict__) #error karena titik_c sudah di free memory

if __name__ == "__main__":
  main()
```

Latihan

Cobalah kode lengkap pada slide sebelumnya dan amati outputnya!

Referensi

Introduction to Computer Science and Programming in Python, MIT
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/lecture-slides-code/MIT6_0001F16_Lec8.pdf

Guttag, John. Introduction to Computation and Programming Using Python: With Application to Understanding Data Second Edition. MIT Press, 2016. ISBN: 9780262529624.