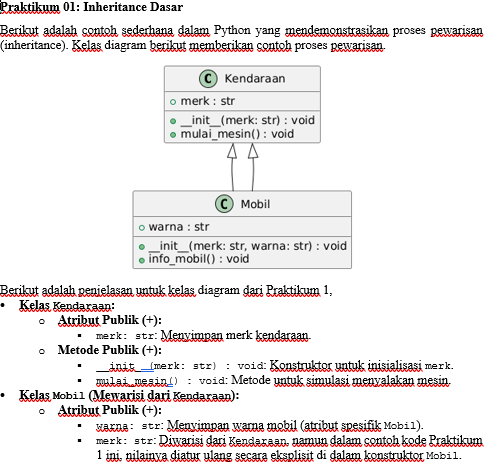
|  |  |
| --- | --- |
|  | **POLITEKNIK NEGERI SEMARANG JURUSAN TEKNIK ELEKTRO**  **PROGRAM STUDI STR TEKNOLOGI REKAYASA KOMPUTER** |

JOBSHEET 04:

MEMAHAMI PEWARISAN (INHERITANCE) DALAM PYTHON

****

# Dosen:

Ir. Prayitno, S.ST., M.T., Ph.D.

# Nama Mahasiswa : NIM Mahasiswa :

ˆ.·.·ç· **Tahun Akademik 2025**

# Tujuan Instruksional Khusus

Setelah menyelesaikan praktikum ini, mahasiswa diharapkan mampu:

* 1. Menjelaskan konsep dasar pewarisan (inheritance) dalam Pemrograman Berorientasi Objek.
  2. Mengidentifikasi hubungan antara kelas induk (Parent Class/Superclass) dan kelas anak (Child Class/Subclass).
  3. Mengimplementasikan pewarisan sederhana (single inheritance) dalam Python.
  4. Menggunakan fungsi super() untuk memanggil metode dari kelas induk di dalam kelas anak.
  5. Menerapkan penggantian metode (method overriding) untuk menyediakan implementasi spesifik di kelas anak.
  6. Menyusun hierarki kelas sederhana menggunakan pewarisan.

# Alat dan Bahan Perangkat Lunak:

* Sistem operasi Windows/Linux/macOS (sesuai ketersediaan).
* Python 3.x (versi terbaru yang stabil).
* IDE atau Code Editor (misalnya, PyCharm, VS Code, atau IDLE).

# Perangkat Keras:

* Komputer/Laptop dengan spesifikasi memadai untuk menjalankan Python.

# Bahan Pendukung:

* Modul/slide perkuliahan yang menjelaskan dasar pemrograman Python.
* Koneksi internet (opsional, untuk referensi tambahan).

# Dasar Teori

Pewarisan (Inheritance) adalah salah satu konsep fundamental dan mekanisme kunci dalam Pemrograman Berorientasi Objek (OOP). Konsep ini memungkinkan sebuah kelas baru, yang disebut kelas anak (Child Class/Subclass), untuk mewarisi atau mengambil alih properti (atribut) dan perilaku (metode) dari kelas yang sudah ada, yang disebut kelas induk (Parent Class/Superclass). Hubungan ini sering digambarkan sebagai relasi "is-a" (adalah sebuah), yang membantu dalam membangun hierarki kelas yang logis dan terstruktur.

* **Kelas Induk (Parent Class / Superclass / Base Class):** Kelas yang fitur-fiturnya (atribut dan metode) diwariskan ke kelas lain. Kelas ini seringkali mendefinisikan antarmuka dan fungsionalitas umum.
* **Kelas Anak (Child Class / Subclass / Derived Class):** Kelas yang mewarisi fitur dari kelas induk. Kelas anak dapat menggunakan fitur yang diwarisi, menambahkan fitur baru yang spesifik untuk dirinya, atau memodifikasi (override) perilaku yang diwarisi.

**Sintaks Dasar Inheritance:** Di Python, pewarisan diimplementasikan dengan mencantumkan nama kelas induk dalam tanda kurung setelah nama kelas anak:

Python

class KelasInduk:

# Definisi kelas induk pass

class KelasAnak(KelasInduk):

# Definisi kelas anak, mewarisi dari KelasInduk pass

**Keuntungan Inheritance:** Manfaat utama yang sering dikaitkan dengan inheritance meliputi:

* **Code Reusability (Penggunaan Ulang Kode):** Inheritance secara luas diakui memfasilitasi penggunaan ulang kode. Atribut dan metode yang umum untuk beberapa kelas dapat didefinisikan sekali di kelas induk dan digunakan kembali oleh semua kelas anaknya, mengurangi redundansi dan potensi error, serta menghemat waktu pengembangan.
* **Organisasi Kode dan Hierarki:** Inheritance membantu mengorganisasi kelas ke dalam hierarki yang mencerminkan hubungan konseptual antar abstraksi. Struktur hierarkis ini dapat membuat sistem lebih mudah dipahami.
* **Extensibility (Kemudahan Perluasan):** Sistem menjadi lebih mudah diperluas karena fungsionalitas baru dapat ditambahkan dengan membuat kelas anak yang mengkhususkan atau menambah fitur kelas induk, seringkali tanpa perlu memodifikasi kode induk yang sudah ada.
* **Polimorfisme:** Inheritance merupakan dasar bagi polimorfisme, di mana objek dari kelas anak dapat diperlakukan sebagai objek dari kelas induknya, memungkinkan fleksibilitas dalam desain perangkat lunak.

Meskipun demikian, beberapa studi dan ahli juga mengingatkan bahwa penggunaan inheritance yang berlebihan atau tidak tepat, terutama dalam hierarki yang dalam atau kompleks, dapat menimbulkan masalah. Hal ini bisa meliputi peningkatan kompleksitas, ketergantungan yang erat antar kelas (coupling), dan potensi masalah dalam pemeliharaan. Beberapa penelitian bahkan menunjukkan bahwa kedalaman hierarki inheritance tidak selalu berkorelasi positif dengan kemudahan pemeliharaan. Oleh karena itu, penting untuk menggunakan inheritance secara bijak dan mempertimbangkan alternatif seperti komposisi objek jika lebih sesuai.

**Method Overriding (Penggantian Metode):** Kelas anak dapat memberikan implementasi spesifik untuk metode yang diwarisi dari kelas induk. Proses ini disebut *method overriding*. Metode yang di-override di kelas anak memiliki nama dan signatur yang sama dengan metode di kelas induk. Ketika metode dipanggil pada objek kelas anak, versi metode dari kelas anak yang akan dieksekusi.

**Fungsi super():** Di Python, super() adalah fungsi bawaan yang memungkinkan kelas anak untuk memanggil metode dari kelas induknya. Ini sangat berguna dalam metode yang di- override (termasuk konstruktor init ) untuk menjalankan logika dari kelas induk sebelum atau sesudah menambahkan logika spesifik kelas anak.

Contoh penggunaan super() dalam init : Python

class KelasInduk:

def init (self, attr\_induk): # Inisialisasi bagian induk self.attr\_induk = attr\_induk

class KelasAnak(KelasInduk):

def init (self, attr\_induk, attr\_anak):

super(). init (attr\_induk) # Memanggil init induk # Inisialisasi bagian anak

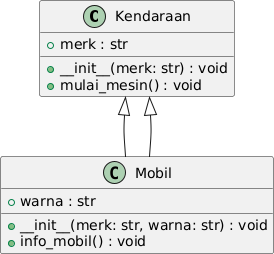
self.attr\_anak = attr\_anak

Secara keseluruhan, inheritance adalah alat yang kuat dalam OOP untuk pemodelan, penggunaan ulang kode, dan perluasan sistem, namun penggunaannya memerlukan

pertimbangan desain yang matang untuk memaksimalkan manfaat dan meminimalkan potensi kekurangannya.

# Langkah Praktikum Praktikum 01: Inheritance Dasar

Berikut adalah contoh sederhana dalam Python yang mendemonstrasikan proses pewarisan (inheritance). Kelas diagram berikut memberikan contoh proses pewarisan.



Berikut adalah penjelasan untuk kelas diagram dari Praktikum 1,

* **Kelas Kendaraan:**

# Atribut Publik (+):

* + - merk: str: Menyimpan merk kendaraan.

# Metode Publik (+):

* + - init (merk: str) : void: Konstruktor untuk inisialisasi merk.
    - mulai\_mesin() : void: Metode untuk simulasi menyalakan mesin.
* **Kelas Mobil (Mewarisi dari Kendaraan):**

# Atribut Publik (+):

* + - warna: str: Menyimpan warna mobil (atribut spesifik Mobil).
    - merk: str: Diwarisi dari Kendaraan, namun dalam contoh kode Praktikum 1 ini, nilainya diatur ulang secara eksplisit di dalam konstruktor Mobil.

# Metode Publik (+):

* + - init (merk: str, warna: str) : void: Konstruktor untuk Mobil, menginisialisasi merk dan warna. (Belum menggunakan super() di praktikum ini).
    - info\_mobil() : void: Metode spesifik untuk menampilkan info mobil. (Metode mulai\_mesin diwarisi dari Kendaraan).

Diagram ini menunjukkan hubungan pewarisan dasar dimana kelas Mobil mewarisi (<|--) dari kelas Kendaraan. Objek Mobil dapat mengakses metode mulai\_mesin yang didefinisikan di Kendaraan.

Kemudian untuk kode praktikum dalam python dapat dilihat sebagai berikut:

01: # Kelas Induk 02: class Kendaraan:

03: def init (self, merk): 04: self.merk = merk

05:

06:

07:

08:

def mulai\_mesin(self):

print(f"Mesin kendaraan {self.merk} dinyalakan.")

09: # Kelas Anak (mewarisi dari Kendaraan) 10: class Mobil(Kendaraan):

11:

12:

13:

14:

15:

16:

17:

18:

def init (self, merk, warna):

# Memanggil init kelas induk (akan dibahas di Praktikum 2) # Untuk sekarang, kita set merk secara manual

self.merk = merk # Mewarisi 'merk' tapi di-set ulang di sini self.warna = warna # Atribut khusus Mobil

def info\_mobil(self):

print(f"Ini adalah mobil {self.merk} berwarna {self.warna}.")

19:

20: # --- Kode Utama ---

21: if name == " main ":

22:

23:

24:

25:

26:

27:

28:

29:

30:

31:

mobil\_tesla = Mobil("Tesla Model S", "Merah")

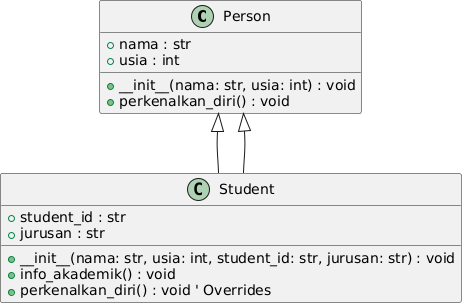
# Memanggil metode dari kelas anak mobil\_tesla.info\_mobil()

# Memanggil metode yang diwarisi dari kelas induk mobil\_tesla.mulai\_mesin()

# Mengakses atribut yang diwarisi (dan di-set di anak) print(f"Merk mobil: {mobil\_tesla.merk}")

# Praktikum 02: Menggunakan super() dalam konstruktor

**Tujuan:** Menggunakan super(). init () untuk inisialisasi atribut kelas induk (Person) dari dalam konstruktor kelas anak (Student).



* **Kelas Person:**
  + **Atribut Publik (+):** nama, usia.
  + **Metode Publik (+):**  init (nama, usia), perkenalkan\_diri().
* **Kelas Student (Mewarisi dari Person):**
  + **Atribut Publik (+):** student\_id, jurusan. (Atribut nama dan usia diwarisi).
  + **Metode Publik (+):**  init (nama, usia, student\_id, jurusan) (memanggil

super(). init ()), info\_akademik(), perkenalkan\_diri() (meng-override metode induk).

Diagram ini menunjukkan kelas Student yang mewarisi dari Person. Fokus utama praktikum ini adalah pada metode init di Student yang menggunakan super(). init () untuk memanggil konstruktor Person, memastikan inisialisasi yang benar dari atribut nama dan usia sebelum menambahkan atribut spesifik Student.

Kode Python:

01: # Kelas Induk 02: class Person:

03:

04:

05:

06:

07:

08:

09:

tahun.")

def init (self, nama, usia):

print(f"(Memanggil init Person untuk '{nama}')") self.nama = nama

self.usia = usia

def perkenalkan\_diri(self):

print(f"Halo, nama saya {self.nama}, usia saya {self.usia}

10:

11: # Kelas Anak (mewarisi dari Person) 12: class Student(Person):

13:

14:

15:

dan usia 16:

def init (self, nama, usia, student\_id, jurusan): print(f"(Memanggil init Student untuk '{nama}')")

# Memanggil init dari kelas Person untuk inisialisasi nama

super(). init (nama, usia)

17: # Menambahkan atribut khusus Student 18: self.student\_id = student\_id

19: self.jurusan = jurusan

20: print(f"(Inisialisasi atribut Student selesai untuk '{nama}')") 21:

22: def info\_akademik(self):

23: print(f"ID Mahasiswa: {self.student\_id}") 24: print(f"Jurusan: {self.jurusan}")

25:

26: # Override perkenalkan\_diri untuk menambahkan info student 27: def perkenalkan\_diri(self):

28: super().perkenalkan\_diri() # Panggil versi Person

29: print(f"Saya adalah mahasiswa dengan ID {self.student\_id}, jurusan {self.jurusan}.")

30:

31: # --- Kode Utama ---

32: if name == " main ":

33: # Membuat objek Person

34: dosen = Person("Pak Anton", 45)

35: print("-" \* 20)

36: dosen.perkenalkan\_diri()

37: print("\n" + "=" \* 30 + "\n")

38:

39: # Membuat objek Student

40: mahasiswa = Student("Dewi", 20, "MHS001", "Teknik Komputer") 41: print("-" \* 20)

42: mahasiswa.perkenalkan\_diri() # Panggil versi override di Student 43: print("-" \* 20)

44: mahasiswa.info\_akademik() # Panggil metode khusus Student 45:

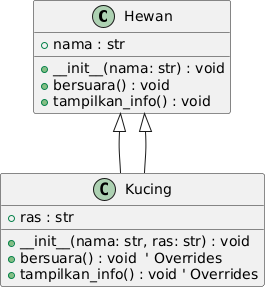
46: # Cek atribut yang diwarisi

47: print(f"\nUsia mahasiswa {mahasiswa.nama}: {mahasiswa.usia}") # usia diwarisi dari Person

**Observasi:** Perhatikan pesan print yang menunjukkan urutan pemanggilan init . Amati bagaimana super(). init (nama, usia) dalam Student memastikan bahwa nama dan usia diinisialisasi oleh konstruktor Person sebelum atribut student\_id dan jurusan ditambahkan. Bandingkan output perkenalkan\_diri() dari objek dosen dan mahasiswa.

# Praktikum 03: Method Overriding dan super()

Tujuan: Mengganti (override) metode dari kelas induk di kelas anak dan menggunakan super() untuk memanggil implementasi asli dari induk.



# Penjelasan Kelas Diagram:

* **Kelas Hewan:**

# Atribut Publik (+):

* + - nama: str: Menyimpan nama hewan.

# Metode Publik (+):

* + - init (nama: str) : void: Konstruktor untuk inisialisasi nama.
    - bersuara() : void: Metode dasar untuk hewan bersuara.
    - tampilkan\_info() : void: Metode dasar untuk menampilkan info hewan.
* **Kelas Kucing (Mewarisi dari Hewan):**

# Atribut Publik (+):

* + - ras: str: Menyimpan ras kucing. (Atribut nama diwarisi).

# Metode Publik (+):

* + - init (nama: str, ras: str) : void: Konstruktor Kucing. Memanggil

super(). init (nama) untuk inisialisasi nama dari Hewan, lalu inisialisasi ras.

* + - bersuara() : void: *Override* metode bersuara dari Hewan untuk memberikan implementasi suara kucing yang spesifik ("Meow!").
    - tampilkan\_info() : void: *Override* metode tampilkan\_info dari Hewan. Metode ini memanggil super().tampilkan\_info() untuk menjalankan implementasi asli dari Hewan, kemudian menambahkan informasi spesifik tentang ras kucing.

Diagram ini menunjukkan bagaimana kelas anak (Kucing) dapat mengganti (override) metode (bersuara, tampilkan\_info) yang diwarisi dari kelas induk (Hewan). Kelas Kucing menyediakan perilakunya sendiri untuk metode-metode ini. Penggunaan super().tampilkan\_info() dalam metode tampilkan\_info yang di-override menunjukkan cara memperluas fungsionalitas induk, bukan hanya menggantinya sepenuhnya.

Kode Program

01: # Kelas Induk 02: class Hewan:

03: def init (self, nama):

04: self.nama = nama 05:

06: def bersuara(self):

07: print(f"{self.nama} mengeluarkan suara generik.") 08:

09: def tampilkan\_info(self):

10: print(f"Ini adalah hewan bernama {self.nama}.") 11:

12: # Kelas Anak

13: class Kucing(Hewan):

14: def init (self, nama, ras):

15: super(). init (nama) # Panggil init induk 16: self.ras = ras

17:

18: # Method Overriding untuk bersuara 19: def bersuara(self):

20: print(f"{self.nama} (Kucing) mengeong: Meow!") 21:

22: # Method Overriding untuk tampilkan\_info, memanggil versi induk 23: def tampilkan\_info(self):

24: super().tampilkan\_info() # Panggil implementasi dari Hewan

25: print(f"Ini adalah kucing ras {self.ras}.") # Tambahkan info spesifik

26:

27: # --- Kode Utama ---

28: if name == " main ":

29: hewan\_umum = Hewan("Makhluk")

30: kucing\_persia = Kucing("Puspus", "Persia") 31:

32: print("Info Hewan Umum:") 33: hewan\_umum.tampilkan\_info()

34: hewan\_umum.bersuara()

35: print("-" \* 20)

36:

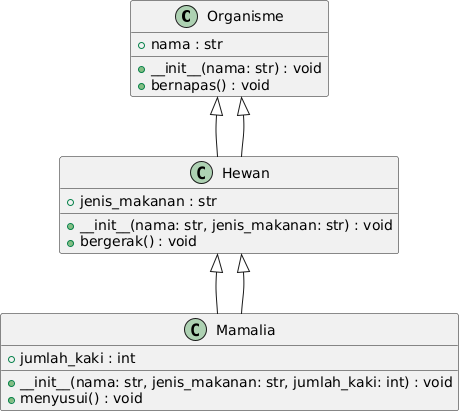
37: print("Info Kucing Persia:")

38: kucing\_persia.tampilkan\_info() # Akan memanggil versi Kucing 39: kucing\_persia.bersuara() # Akan memanggil versi Kucing

**Observasi:** Bandingkan output dari bersuara() dan tampilkan\_info() saat dipanggil pada objek hewan\_umum dan kucing\_persia. Perhatikan bagaimana super().tampilkan\_info() di dalam Kucing.tampilkan\_info() memungkinkan penambahan fungsionalitas tanpa menghilangkan fungsionalitas asli dari induk.

# Praktikum 04: Multi-level Inheritance (Pewarisan Bertingkat)

**Tujuan:** Memahami bagaimana sebuah kelas dapat mewarisi dari kelas anak lain, membentuk rantai pewarisan (A -> B -> C).



* **Kelas Organisme:**

# Atribut Publik (+):

* + - nama: str: Menyimpan nama organisme.

# Metode Publik (+):

* + - init (nama: str) : void: Konstruktor untuk inisialisasi nama.
    - bernapas() : void: Metode untuk simulasi pernapasan.
* **Kelas Hewan (Mewarisi dari Organisme):**

# Atribut Publik (+):

* + - jenis\_makanan: str: Menyimpan jenis makanan hewan. (Atribut nama

diwarisi dari Organisme).

# Metode Publik (+):

* + - init (nama: str, jenis\_makanan: str) : void: Konstruktor, memanggil super(). init () untuk inisialisasi nama dari Organisme, lalu inisialisasi jenis\_makanan.
    - bergerak() : void: Metode untuk simulasi pergerakan. (Metode bernapas

diwarisi).

* **Kelas Mamalia (Mewarisi dari Hewan):**

# Atribut Publik (+):

* + - jumlah\_kaki: int: Menyimpan jumlah kaki mamalia. (Atribut nama dan

jenis\_makanan diwarisi).

# Metode Publik (+):

* + - init (nama: str, jenis\_makanan: str, jumlah\_kaki: int) : void: Konstruktor, memanggil super(). init () untuk inisialisasi nama dan jenis\_makanan dari Hewan, lalu inisialisasi jumlah\_kaki.
    - menyusui() : void: Metode spesifik untuk mamalia. (Metode bernapas dan

bergerak diwarisi).

Diagram ini menunjukkan hierarki pewarisan bertingkat (Organisme <- Hewan <- Mamalia), di mana Mamalia mewarisi fitur dari Hewan, yang pada gilirannya mewarisi fitur dari Organisme. Panah (<|--) menunjukkan arah pewarisan (dari anak ke induk).

Kode Program:

01: # Kelas Induk Level 1

02: class Organisme:

03: def init (self, nama):

04: self.nama = nama

05: print(f"Organisme '{self.nama}' diciptakan.") 06:

07: def bernapas(self):

08: print(f"{self.nama} sedang bernapas.") 09:

10: # Kelas Anak Level 1 (mewarisi dari Organisme) 11: class Hewan(Organisme):

12: def init (self, nama, jenis\_makanan):

13: super(). init (nama) # Panggil init Organisme 14: self.jenis\_makanan = jenis\_makanan

15: print(f"Hewan '{self.nama}' adalah {self.jenis\_makanan}.") 16:

17: def bergerak(self):

18: print(f"{self.nama} sedang bergerak.") 19:

20: # Kelas Anak Level 2 (mewarisi dari Hewan) 21: class Mamalia(Hewan):

22: def init (self, nama, jenis\_makanan, jumlah\_kaki):

23: super(). init (nama, jenis\_makanan) # Panggil init Hewan 24: self.jumlah\_kaki = jumlah\_kaki

25: print(f"Mamalia '{self.nama}' memiliki {self.jumlah\_kaki} kaki.") 26:

27: def menyusui(self):

28: print(f"{self.nama} sedang menyusui.") 29:

30: # --- Kode Utama ---

31: if name == " main ":

32: kucing = Mamalia("Kucing Persia", "Karnivora", 4) 33: print("-" \* 20)

34:

35: # Memanggil metode dari kelas Mamalia 36: kucing.menyusui()

37:

38: # Memanggil metode yang diwarisi dari kelas Hewan 39: kucing.bergerak()

40: print(f"Jenis makanan: {kucing.jenis\_makanan}") 41:

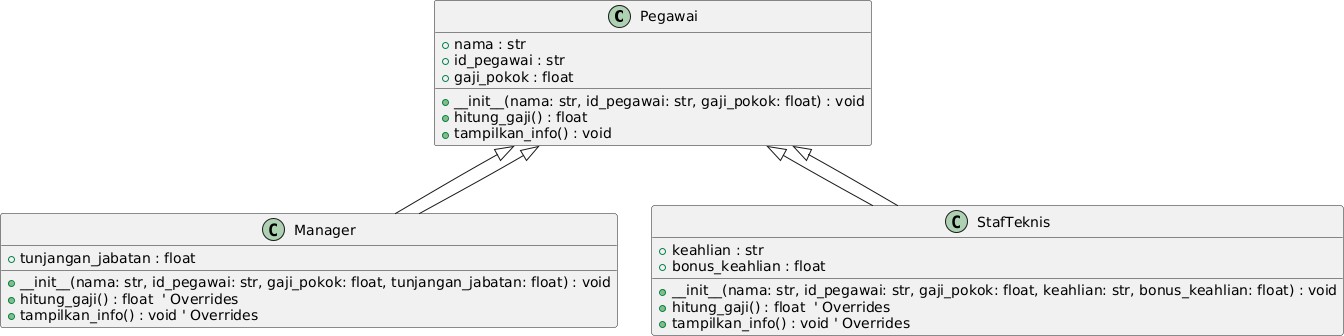
42: # Memanggil metode yang diwarisi dari kelas Organisme 43: kucing.bernapas()

44: print(f"Nama organisme: {kucing.nama}")

**Observasi:** Perhatikan bagaimana objek kucing (dari kelas Mamalia) dapat mengakses metode dan atribut dari kelas Hewan dan Organisme karena adanya rantai pewarisan. Lacak urutan pemanggilan konstruktor ( init ) melalui super().

# Praktikum 05: Hierarchical Inheritance

Tujuan: Mendemonstrasikan bagaimana satu kelas induk (Pegawai) dapat diwarisi oleh beberapa kelas anak yang berbeda (Manager, StafTeknis) untuk merepresentasikan spesialisasi yang berbeda dari konsep dasar yang sama.



* **Kelas Pegawai:**
  + **Atribut Publik (+):** nama, id\_pegawai, gaji\_pokok.
  + **Metode Publik (+):** init (), hitung\_gaji() (versi dasar), tampilkan\_info() (versi dasar).
* **Kelas Manager (Mewarisi dari Pegawai):**
  + **Atribut Publik (+):** tunjangan\_jabatan (Atribut lain diwarisi).
  + **Metode Publik (+):** init () (memanggil super), hitung\_gaji() (override),

tampilkan\_info() (override, memanggil super).

* **Kelas StafTeknis (Mewarisi dari Pegawai):**
  + **Atribut Publik (+):** keahlian, bonus\_keahlian (Atribut lain diwarisi).
  + **Metode Publik (+):** init () (memanggil super), hitung\_gaji() (override),

tampilkan\_info() (override, memanggil super).

Diagram ini menunjukkan kelas Pegawai sebagai kelas induk, dan Manager serta StafTeknis sebagai kelas anak yang mewarisi langsung dari Pegawai, menggambarkan struktur pewarisan hierarkis.

Kode Program:

01: import locale

02:

03: # Set locale ke Indonesia untuk format mata uang 04: try:

05: locale.setlocale(locale.LC\_ALL, 'id\_ID.UTF-8')

06: except locale.Error:

07: print("Locale id\_ID.UTF-8 tidak tersedia, menggunakan locale default.")

08:

09: # Fungsi helper untuk format mata uang 10: def format\_rupiah(angka):

11: return locale.currency(angka, grouping=True, symbol='Rp ') 12:

13: # Kelas Induk 14: class Pegawai:

15:

16:

17:

18:

19:

20:

21:

22:

23:

24:

25:

def init (self, nama, id\_pegawai, gaji\_pokok): self.nama = nama

self.id\_pegawai = id\_pegawai self.gaji\_pokok = gaji\_pokok

def hitung\_gaji(self):

# Gaji dasar, bisa di-override return self.gaji\_pokok

def tampilkan\_info(self):

print(f"ID: {self.id\_pegawai}, Nama: {self.nama}")

26: print(f" Gaji Pokok: {format\_rupiah(self.gaji\_pokok)}") 27:

28: # Kelas Anak 1

29: class Manager(Pegawai):

30: def init (self, nama, id\_pegawai, gaji\_pokok, tunjangan\_jabatan):

31: super(). init (nama, id\_pegawai, gaji\_pokok)

32: self.tunjangan\_jabatan = tunjangan\_jabatan 33:

34: # Override hitung\_gaji 35: def hitung\_gaji(self):

36: gaji\_total = super().hitung\_gaji() + self.tunjangan\_jabatan 37: return gaji\_total

38:

39: # Override tampilkan\_info (memanggil versi induk) 40: def tampilkan\_info(self):

41: print("--- Info Manager ---")

42: super().tampilkan\_info() # Panggil info dasar dari Pegawai

43: print(f" Tunjangan Jabatan:

{format\_rupiah(self.tunjangan\_jabatan)}")

44: print(f" Total Gaji: {format\_rupiah(self.hitung\_gaji())}") 45:

46: # Kelas Anak 2

47: class StafTeknis(Pegawai):

48: def init (self, nama, id\_pegawai, gaji\_pokok, keahlian, bonus\_keahlian):

49: super(). init (nama, id\_pegawai, gaji\_pokok) 50: self.keahlian = keahlian

51: self.bonus\_keahlian = bonus\_keahlian 52:

53: # Override hitung\_gaji 54: def hitung\_gaji(self):

55: gaji\_total = super().hitung\_gaji() + self.bonus\_keahlian 56: return gaji\_total

57:

58: # Override tampilkan\_info (memanggil versi induk) 59: def tampilkan\_info(self):

60: print("--- Info Staf Teknis ---")

61: super().tampilkan\_info() # Panggil info dasar dari Pegawai 62: print(f" Keahlian: {self.keahlian}")

63: print(f" Bonus Keahlian: {format\_rupiah(self.bonus\_keahlian)}") 64: print(f" Total Gaji: {format\_rupiah(self.hitung\_gaji())}")

65:

66:

67: # --- Kode Utama ---

68: if name == " main ":

69: manager1 = Manager("Budi Santoso", "M001", 10000000, 5000000)

70: staf1 = StafTeknis("Citra Lestari", "S001", 7000000, "Python Programming", 1500000)

71: pegawai\_baru = Pegawai("Rian", "P005", 5000000) # Pegawai biasa 72:

73: print("Menampilkan Informasi Pegawai:") 74: manager1.tampilkan\_info()

75: print("-" \* 30)

76: staf1.tampilkan\_info()

77: print("-" \* 30)

78:

79: print("--- Info Pegawai Baru ---") 80: pegawai\_baru.tampilkan\_info()

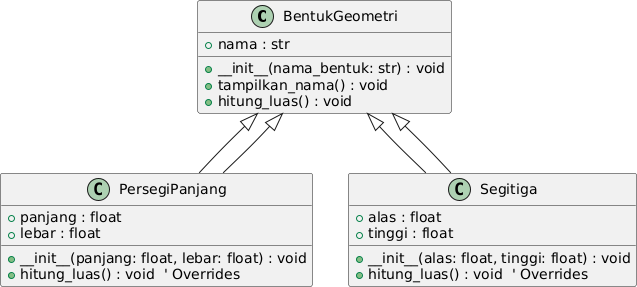
81: print(f" Total Gaji: {format\_rupiah(pegawai\_baru.hitung\_gaji())}") # Memanggil versi Pegawai

82: print("-" \* 30)

**Observasi:** Perhatikan bagaimana kelas Manager dan StafTeknis keduanya mewarisi dari kelas Pegawai yang sama. Masing-masing menambahkan atribut spesifik (tunjangan\_jabatan untuk Manager, keahlian dan bonus\_keahlian untuk StafTeknis) dan meng-override metode hitung\_gaji dan tampilkan\_info untuk mencerminkan peran mereka yang berbeda, sambil tetap memanfaatkan fungsionalitas dasar dari Pegawai melalui super().

# Praktikum 06: Memeriksa Tipe Instance dengan instance()

Tujuan: Menggunakan fungsi isinstance() untuk memeriksa apakah sebuah objek merupakan instance dari kelas tertentu atau dari kelas turunannya dalam hierarki pewarisan.



# Penjelasan Kelas Diagram:

* **Kelas BentukGeometri:**

# Atribut Publik (+):

* + - nama: str: Menyimpan nama bentuk geometri.

# Metode Publik (+):

* + - init (nama\_bentuk: str) : void: Konstruktor untuk inisialisasi nama.
    - tampilkan\_nama() : void: Metode untuk menampilkan nama bentuk.
    - hitung\_luas() : void: Metode dasar (generik) untuk menghitung luas, diharapkan di-override oleh kelas anak.
* **Kelas PersegiPanjang (Mewarisi dari BentukGeometri):**

# Atribut Publik (+):

* + - panjang: float: Menyimpan panjang persegi panjang.
    - lebar: float: Menyimpan lebar persegi panjang. (Atribut nama diwarisi).

# Metode Publik (+):

* + - init (panjang: float, lebar: float) : void: Konstruktor, memanggil

super(). init () untuk set nama spesifik, lalu inisialisasi panjang dan lebar.

* + - hitung\_luas() : void: *Override* metode induk untuk memberikan implementasi perhitungan luas persegi panjang. (Metode tampilkan\_nama diwarisi).
* **Kelas Segitiga (Mewarisi dari BentukGeometri):**

# Atribut Publik (+):

* + - alas: float: Menyimpan alas segitiga.
    - tinggi: float: Menyimpan tinggi segitiga. (Atribut nama diwarisi).

# Metode Publik (+):

* + - init (alas: float, tinggi: float) : void: Konstruktor, memanggil super(). init ()

untuk set nama spesifik, lalu inisialisasi alas dan tinggi.

* + - hitung\_luas() : void: *Override* metode induk untuk memberikan implementasi perhitungan luas segitiga. (Metode tampilkan\_nama diwarisi).

Diagram ini identik dengan diagram untuk Praktikum 5 karena menggunakan struktur kelas yang sama. Praktikum 6 berfokus pada penggunaan fungsi isinstance() *di luar* definisi kelas untuk memeriksa tipe objek yang dibuat dari hierarki kelas ini. Fungsi proses\_bentuk() yang menggunakan isinstance() tidak termasuk dalam diagram kelas karena ia bukan metode dari kelas-kelas tersebut.

Kode Program:

01: class BentukGeometri:

02: def init (self, nama\_bentuk):

03: self.nama = nama\_bentuk 04: def tampilkan\_nama(self):

05: print(f"Ini adalah bentuk: {self.nama}") 06: def hitung\_luas(self):

07: print(f"Luas {self.nama} tidak dapat dihitung (generik).") 08:

09: class PersegiPanjang(BentukGeometri):

10: def init (self, panjang, lebar):

11: super(). init ("Persegi Panjang") 12: self.panjang = panjang

13: self.lebar = lebar 14: def hitung\_luas(self):

15: luas = self.panjang \* self.lebar 16: print(f"Luas {self.nama}: {luas}") 17:

18: class Segitiga(BentukGeometri):

19: def init (self, alas, tinggi):

20: super(). init ("Segitiga")

21: self.alas = alas

22: self.tinggi = tinggi 23: def hitung\_luas(self):

24: luas = 0.5 \* self.alas \* self.tinggi 25: print(f"Luas {self.nama}: {luas}") 26:

27: def proses\_bentuk(bentuk):

28: print(f"\nMemproses objek: {type(bentuk)}") 29:

30: if isinstance(bentuk, BentukGeometri):

31: bentuk.tampilkan\_nama()

32: print("Objek ini adalah instance dari BentukGeometri (atau turunannya).")

33:

34: if isinstance(bentuk, PersegiPanjang):

35: print("-> Objek ini juga instance dari PersegiPanjang.") 36: elif isinstance(bentuk, Segitiga):

37: print("-> Objek ini juga instance dari Segitiga.") 38: else:

39: print("Objek ini BUKAN instance dari BentukGeometri.") 40:

41: if isinstance(bentuk, str):

42: print("Objek ini adalah string.") 43:

44: # --- Kode Utama ---

45: if name == " main ": 46: pp = PersegiPanjang(4, 3)

47: seg = Segitiga(5, 2)

48:

49:

50:

51:

52:

53:

54:

bg = BentukGeometri("Lingkaran") # Contoh lain teks = "Ini string"

daftar\_objek = [pp, seg, bg, teks]

for obj in daftar\_objek: proses\_bentuk(obj)

**Observasi:** Perhatikan output dari isinstance(). isinstance(pp, PersegiPanjang) dan isinstance(pp, BentukGeometri) keduanya True. isinstance(bg, BentukGeometri) True, tetapi isinstance(bg, PersegiPanjang) False. isinstance(teks, BentukGeometri) False. Pahami bagaimana isinstance() memeriksa seluruh rantai pewarisan.

# Hasil Praktikum

Lengkapi hasil tabel praktikum berikut:

|  |  |  |
| --- | --- | --- |
| **No.** | **Nama Praktikum** | **Hasil Praktikum** |
| 1 | Praktikum 1: Inheritance Dasar |  |
| 2 | Praktikum 2: Menggunakan super() dalam Konstruktor |  |
| 3 | Praktikum 3: Method Overriding dan super() |  |
| 4 | Praktikum 4: Multi-level Inheritance (Pewarisan Bertingkat) |  |
| 5 | Praktikum 5: Hierarchical Inheritance (Pewarisan Hierarkis) |  |
| 6 | Praktikum 6: Memeriksa Tipe Instance dengan isinstance() |  |

# Penugasan

* 1. Kumpulkan Laporan Praktikum dari jobsheet ini dalam bentuk Microsoft word sesuai dengan format jobsheet praktikum dan dikumpulkan di web LMS. (JANGAN DALAM BENTUK PDF)
  2. Kumpulkan luaran kode praktikum dalam bentuk ipynb yang sudah diunggah pada akun github masing-masing. Lampirkan tautan github yang sudah di unggah melalui laman LMS.
  3. **Buat Program Hierarki Kelas Mahasiswa:** Implementasikan hierarki kelas dalam Python serta **KELAS DIAGRAM** sesuai dengan spesifikasi berikut:
     + **Kelas Induk: Mahasiswa**
       - Memiliki atribut: nama (string), nim (string), ipk (float).
       - Memiliki konstruktor ( init ) yang menerima nama, nim, dan ipk untuk inisialisasi atribut.
       - Memiliki metode tampilkan\_info(self) yang mencetak informasi dasar mahasiswa (nama, nim, ipk). Contoh output:
       - Nama: [Nama Mahasiswa]
       - NIM: [NIM Mahasiswa]
       - IPK: [IPK Mahasiswa]
       - Memiliki metode hitung\_predikat(self) yang mencetak predikat umum berdasarkan IPK (contoh sederhana, Anda boleh memodifikasi rentangnya):
         * IPK >= 3.5: "Dengan Pujian"
         * IPK >= 3.0: "Sangat Memuaskan"
         * IPK >= 2.5: "Memuaskan"
         * IPK < 2.5: "Cukup"
     + **Kelas Anak 1: MahasiswaSarjana** (Mewarisi dari Mahasiswa)
       - Memiliki atribut tambahan: semester (integer), sks\_lulus (integer).
       - Memiliki konstruktor ( init ) yang menerima nama, nim, ipk, semester, dan sks\_lulus. Gunakan super(). init () untuk meneruskan nama, nim, dan ipk ke konstruktor kelas Mahasiswa.
       - **Override** metode tampilkan\_info(self): Panggil dulu super().tampilkan\_info() untuk menampilkan info dasar, lalu tambahkan informasi Semester dan SKS Lulus. Contoh output tambahan:
       - Semester: [Semester]
       - SKS Lulus: [Jumlah SKS]
     + **Kelas Anak 2: MahasiswaMagister** (Mewarisi dari Mahasiswa)
       - Memiliki atribut tambahan: judul\_tesis (string), nama\_pembimbing (string).
       - Memiliki konstruktor ( init ) yang menerima nama, nim, ipk, judul\_tesis, dan

nama\_pembimbing. Gunakan super(). init () untuk meneruskan nama, nim, dan ipk.

* + - * **Override** metode tampilkan\_info(self): Panggil dulu super().tampilkan\_info(), lalu tambahkan informasi Judul Tesis dan Pembimbing. Contoh output tambahan:
      * Judul Tesis: [Judul Tesis Mahasiswa]
      * Pembimbing: [Nama Pembimbing]
      * *(Opsional)* Anda bisa meng-override hitung\_predikat jika ada aturan predikat yang berbeda untuk Magister.

# Kesimpulan

Praktikum ini telah memperkenalkan konsep fundamental pewarisan (inheritance) dalam Pemrograman Berorientasi Objek (OOP) menggunakan Python, yang memungkinkan kelas anak (subclass) untuk mewarisi dan memperluas fungsionalitas dari kelas induk (superclass). Melalui implementasi pewarisan dasar, penggunaan super() untuk memanggil konstruktor dan metode induk, serta penerapan method overriding untuk spesialisasi perilaku, mahasiswa dapat memahami bagaimana inheritance memfasilitasi penggunaan ulang kode (code reusability), menciptakan hierarki kelas yang logis, dan membangun perangkat lunak yang lebih modular, terstruktur, serta mudah dikembangkan lebih lanjut.

# Daftar Pustaka

* 1. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
  2. Guttag, J. V. (2016). *Introduction to Computation and Programming Using Python*. MIT Press.
  3. Python Software Foundation. *Python 3 Documentation*. <https://docs.python.org/3/>